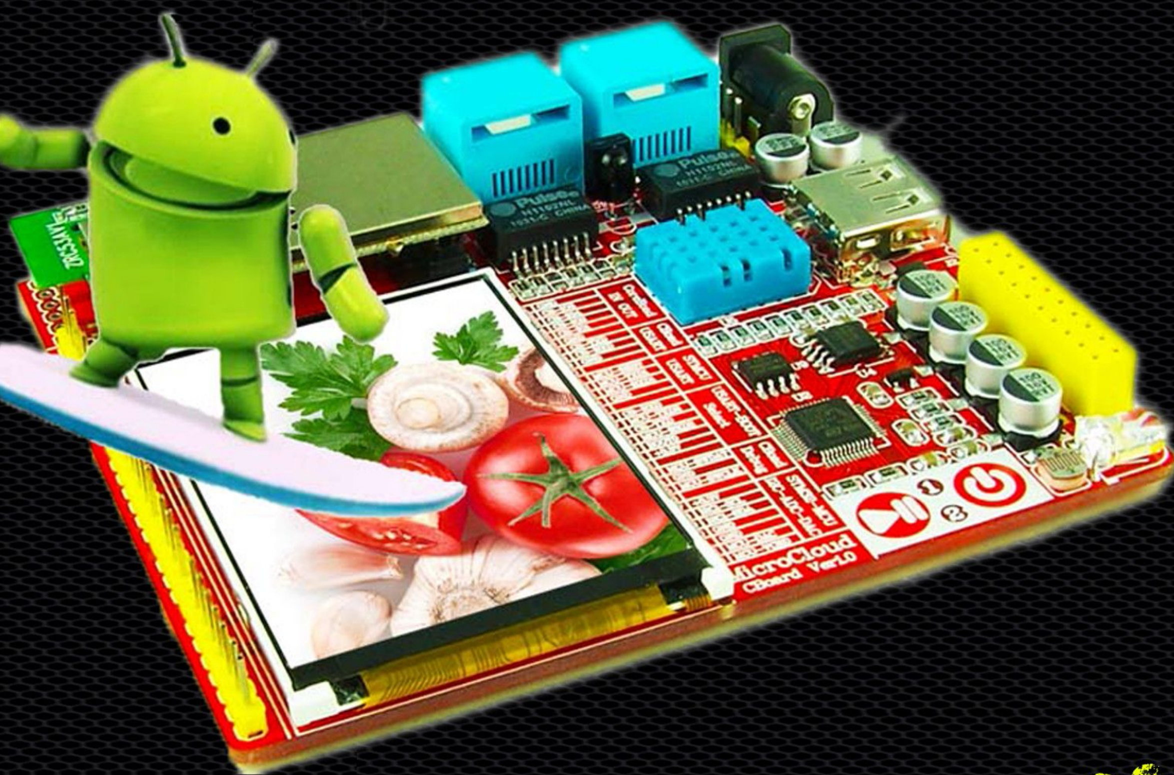
微云电子

Micro cloud

STM32+LINUX

开发板应用手册



首款

带安卓app可远程控制的开发板

一板在手 应有尽有



目 录









STM32+LINUX 开发板简介篇

第 1 章 微云电子 STM32+Linux 开发板资源简介



目前，智能手机移动平台广泛应用，3G/4G 速度快读提升、资费大幅下降，云应用快速展开，与之相应的远程智能控制、广域数据采集、无线传感网、物联网等将得以全面发展，其中关键技术就是计算机网络通信和嵌入式 Linux 系统，仅仅会单片机技术已经远远满足不了后续而来的信息大爆发的浪潮，当下无线 wifi 模块、智能家居等诸多产品的热卖已经暗示着信息浪潮即将到来，了解最新信息动态，掌握自己的未来，已经显得刻不容缓。

STM32+Linux 开发学习板，是 STM32 和网络处理器组成的双处理器平台，STM32 单片机以 C 语言库函数、网络处理平台以 Linux 的标准 c 或 python 等语言并行运行。STM32 单片机平台以高性能多资源的 STM32F051 为主，附带众多片内外外设，以实际应用项目为原型，选取经典软硬件方案，可完成 STM32 单片机深度开发学习。网络处理器与 Linux 平台，主要是面向家庭和个人的微型信息服务中心开发组合，标配有高清摄像头、HIFI 多声道声卡、蓝牙模块、高保真音箱、红外遥控插排及各类调试仿真工具；可用于智能家居、安防监控、物联网信息中心、家庭保健医疗、个人云存储、广域云存储连接、混合云应用、异地 E 家信息中心、家庭娱乐音视频推送等项目学习开发.

1.1 首款带安卓体验控制界面 APP 的 STM32 开发板



微云电子的 STM32 开发板带有 android 手机客户端的 APP 软件，我们可以将 APP 快捷的安装到手机，应用手机客户端和开发板进行通信，进而通过 wifi 控制开发板进行各种操作，如摄像头，液晶屏等。安卓版本的 APP 源码，以及 APP 开发教程，使我们不仅能体验 APP 给带来的便捷，同时还可以学习编写的自己的客户端软件，一块 STM32 开发板使我们达到一学多得的目的。

1.2 带高清 HIFI 音频流媒体服务的 STM32 开发板



与普通带 VS1005 音频解码模块的 STM32 开发板不同，微云开发板带有一体化 Linux 模块，其内含高清数字音频解码器，附带板载高容量网络存储（内含很多优美的音乐、精彩的脱口秀、搞笑的相声、经典的影视英文等音频文件），通过 wifi，手机或者网页等可远程控制诸多音源的切换，停止和播放。开发板还附赠高保真音响，清晰流畅的音频节目能让您学习之余体验震撼的音乐效果。当然我们还可以自己学习尝试修改源代码，以及增添删减音频文件，编辑属于自己个性化的音乐内容。

1.3带高清流媒视频源的 STM32 开发板



微云开发板带有一体化 Linux 模块，其内含高清数字视频流媒体服务器，附带板载高容量网络存储（内含多部高清影视频文件），在学习之余，可以通过局域网内无线 wifi，应用自己的手机或者平板，欣赏当下流行高清电影。同样，我们可以通过学习来尝试修改流媒体服务的程序，打造个性化的影视中心。

1.4云技术信息中心



微云开发板是一个小型的云服务信息中心，可以通过各种终端比如手机、PC 等，获取远程传感器的实时数据（如温湿度，光强等）。同时开发板还具有私有云存储功能，可存储海量的家庭服务信息，如家庭保健医疗，异地 E 家信息等。当然也可以通过学习打造适合自己的特色服务中心，如个性数字图书馆等。

1.5智能控制中心



微云电子的 STM32+Linux 开发板还是一个智能控制中心，我们可以通过手机，电脑等终端在家里控制家电，比如照明灯的通断，红外遥控设备的开启与关闭，遥控电源插排的开关等。通过学习我们可以为自己建造一个超现代化的舒适智能家庭。

1.6Web 型网络监控与图片推送



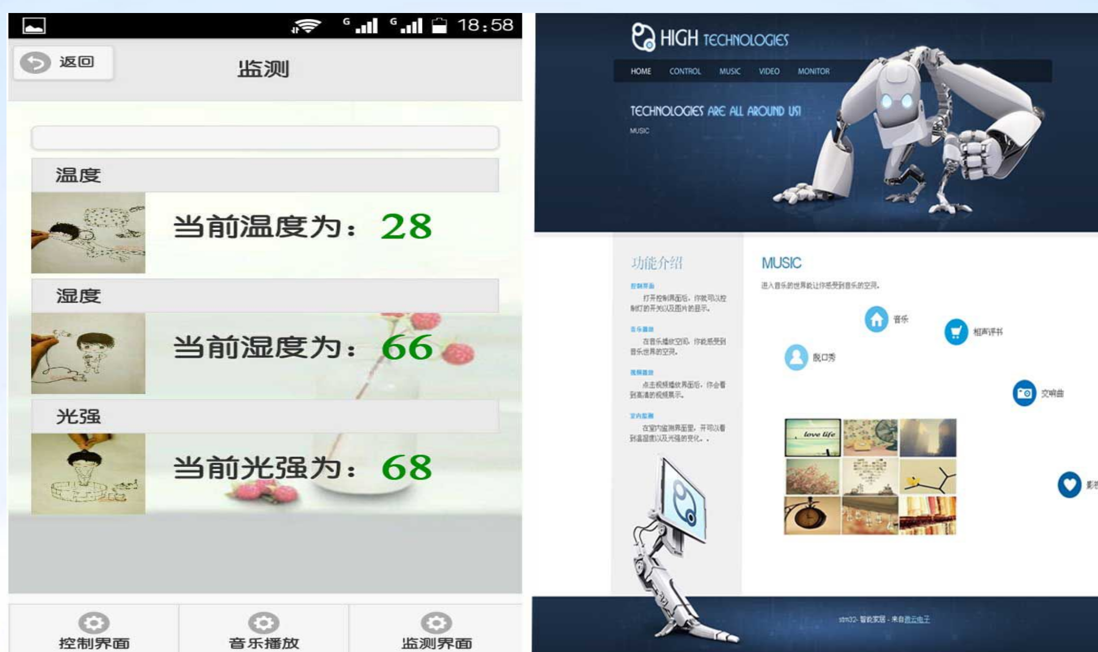
开发板附带高清摄像头，支持 Web 监控和网络推送，可通过 wifi 网络查看摄像头监控画面，并自动获取摄像头截取画面，通过 FTP 上传至个人的计算机或者手机中，以便实时浏览。

1.7室内环境数据的监测中心



开发板带有各种测量室内环境参数的传感器，可以测量室内的温度、湿度和光强，可通过网络利用各种终端设备实时采集室内环境的参数在线监测，使我们随时随地随心的查看自己家里的环境状况。

1.8私有云端 Web 网页和安卓手机 App



开发板内建多种服务器，包括 Web 服务，能构建私有信息中心，炫酷的 Web 网页和便捷的安卓客户控制端能让您随心所欲的掌控一切，享受高科技带来的乐趣。开放的源代码能让您快速打造个性化的私人信息中心。



1.9学习资源与开发辅导

开发板具有详尽完善的开发文档，所有的工程源代码完全开放，特别是专业的售后服务群，多位参与开发的技术人员实时在线进行技术辅导，免除我们的学习后顾之忧，多种套餐组合能满足不同需求。

1.10 学习之忧—开发板学习难度解惑

1.11 开发板及附配件组合简介

STM32+Linux 开发板配备的配件主要有：

- ❖ STM32+Linux 开发板
- ❖ 蓝悦 HIFI 音箱
- ❖ 罗技高清摄像头
- ❖ +5V 电源适配器
- ❖ 网络数据连接线
- ❖ USB 扩展口 4 口 HUB
- ❖ 大容量网络存储 U 盘（**内含系统文件，请勿格式化**）
- ❖ 红外迷你遥控器
- ❖ CH340 USB 转 UART 模块
- ❖ USB 免驱动声卡



开发板的淘宝链接：

http://item.taobao.com/item.htm?spm=a230r.7195193.1997079397.28.Qcm2MO&id=38438758959

除此之外，微云电子开发板还附有增强型安卓套餐，在基础版本上添加了红外插排和 ST-LINK，软件资源上增加了 android、iOS、winphone 等移动平台的联合应用开发案例：

淘宝链接为：

http://meal.taobao.com/mealDetail.htm?spm=2013.1.1000371.d9.7vfQBc&meal_id=132706629&item_num_id=38438758959&seller_id=1037813321

第 2 章 STM32+linux 开发板资源测试体验

2.1局域网内 wifi 网络 web 控制

STM32_LINUX 开发板拿到手之后，按照下图连接好 U 盘，HUB，声卡，摄像头：



给开发板上电，等待约 30s 左右，可以听到开机的音乐，声卡的指示灯在闪烁：PC 笔记本开启无线搜到 ssid 名为 microcloud 的无线 wifi，密码为 12345678，连接。（此处一定要设置电脑自动获取 IP）



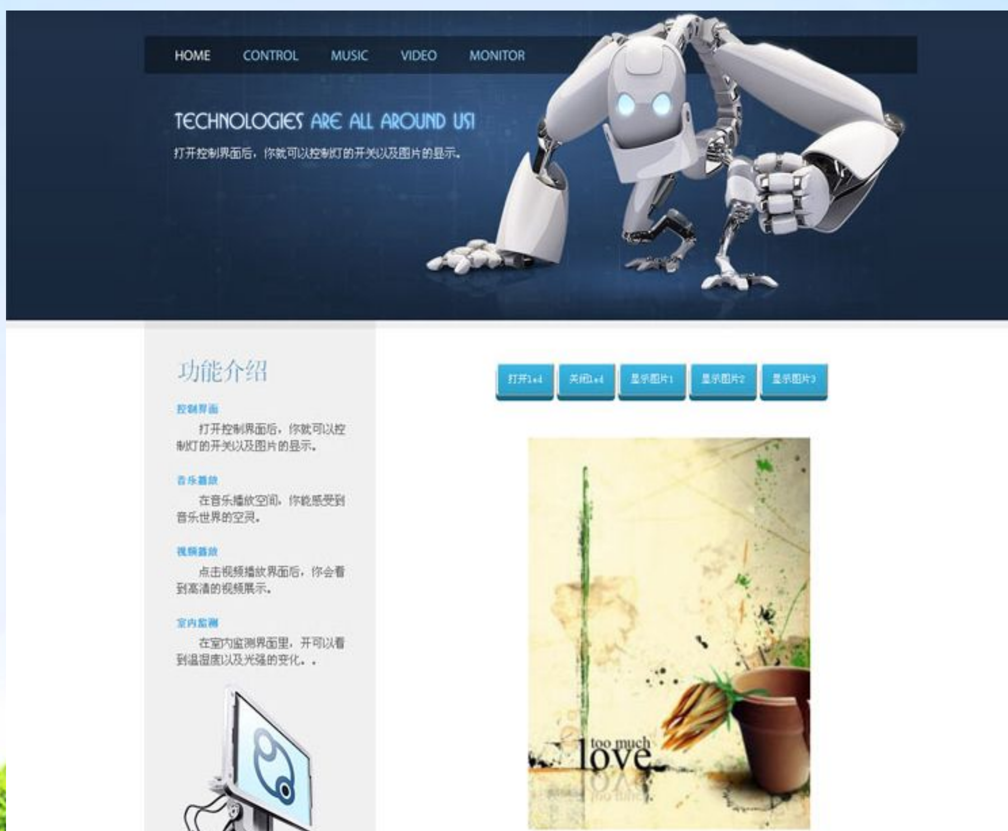
打开火狐浏览器或谷歌浏览器（其他 IE 浏览器不支持视频解码），在浏览器的地



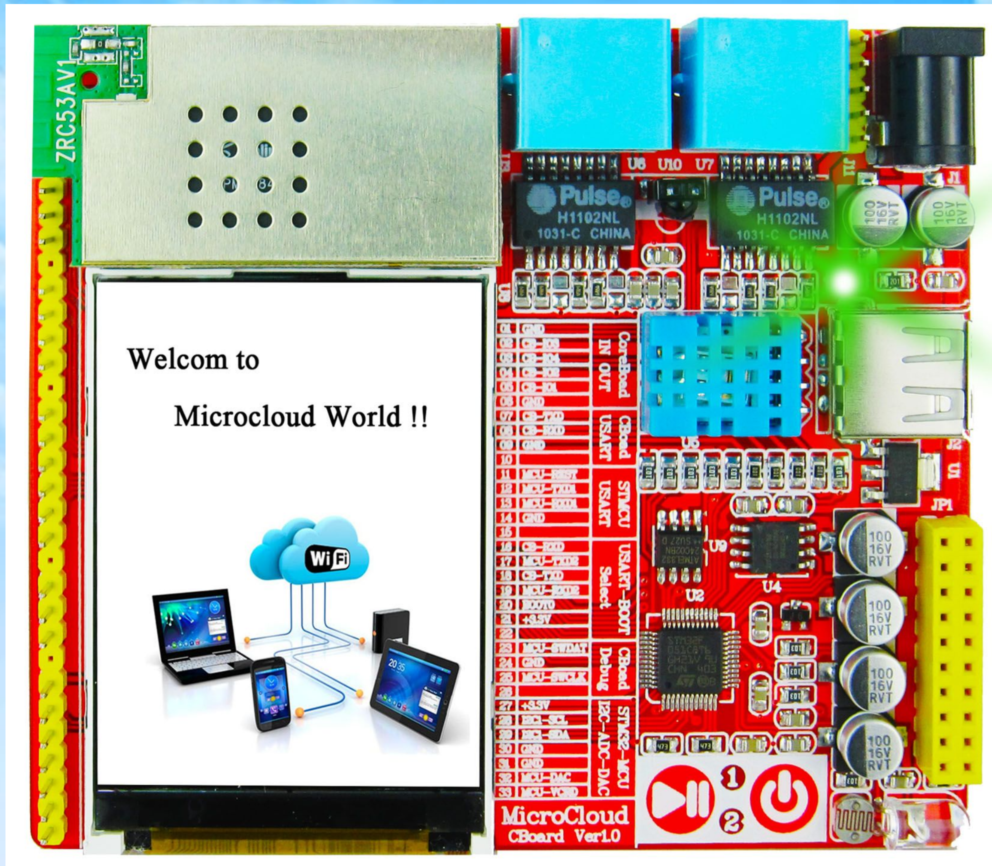
址一栏输入 192.168.1.1，弹出如下的控制界面：



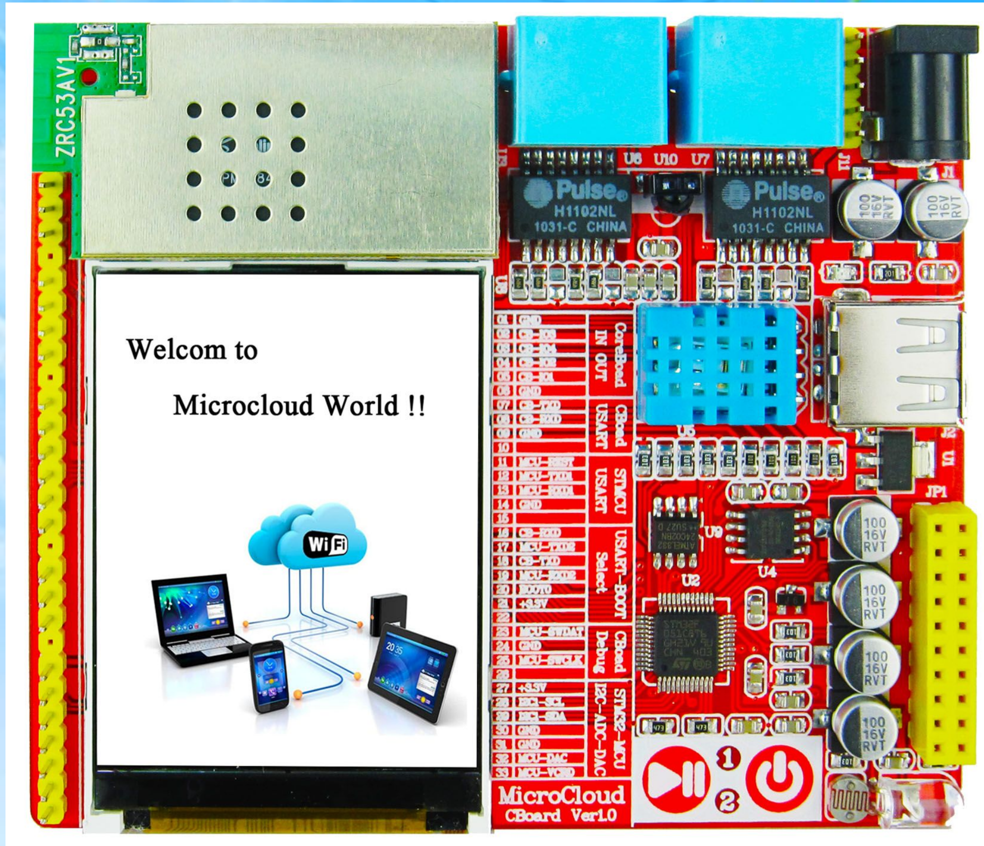
点击 CONTROL 按钮，跳转到控制界面，在这个界面可以控制 led 灯，和液晶屏画面的切换。



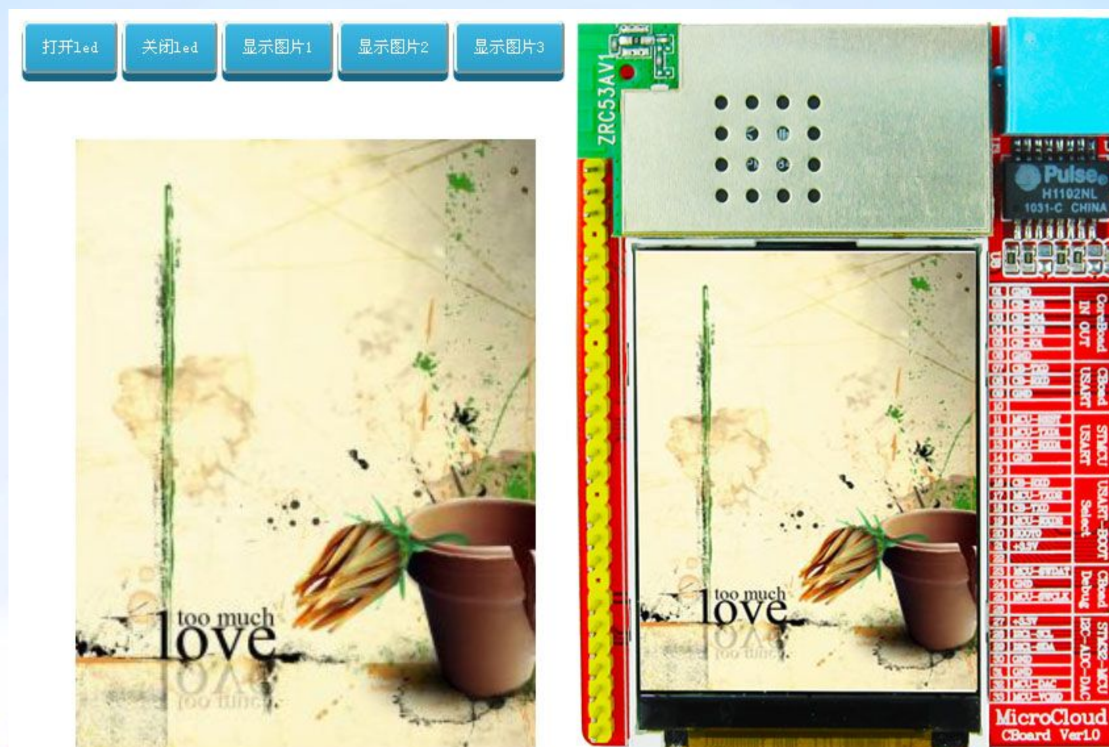
点击打开 led，可以看到开发板的灯被点亮：



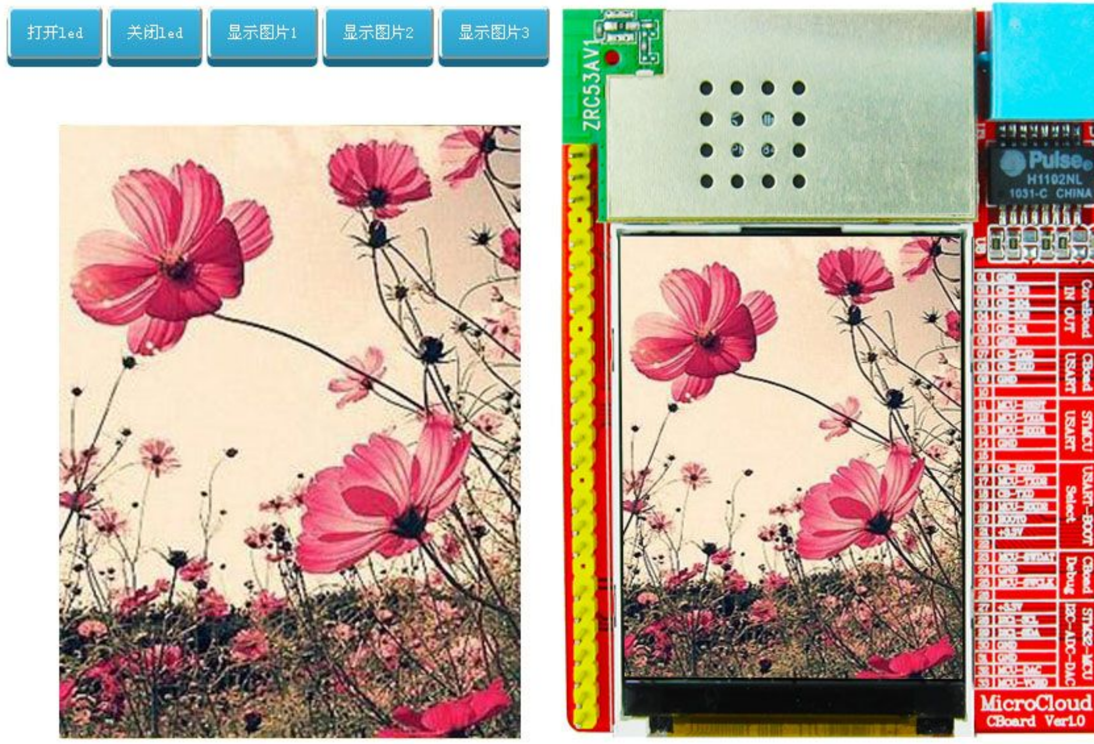
点击关闭 led，可以看到开发板的灯被关闭：



点击显示图片 1，可以看到浏览器的图片切换了，同时液晶屏也在刷图片：

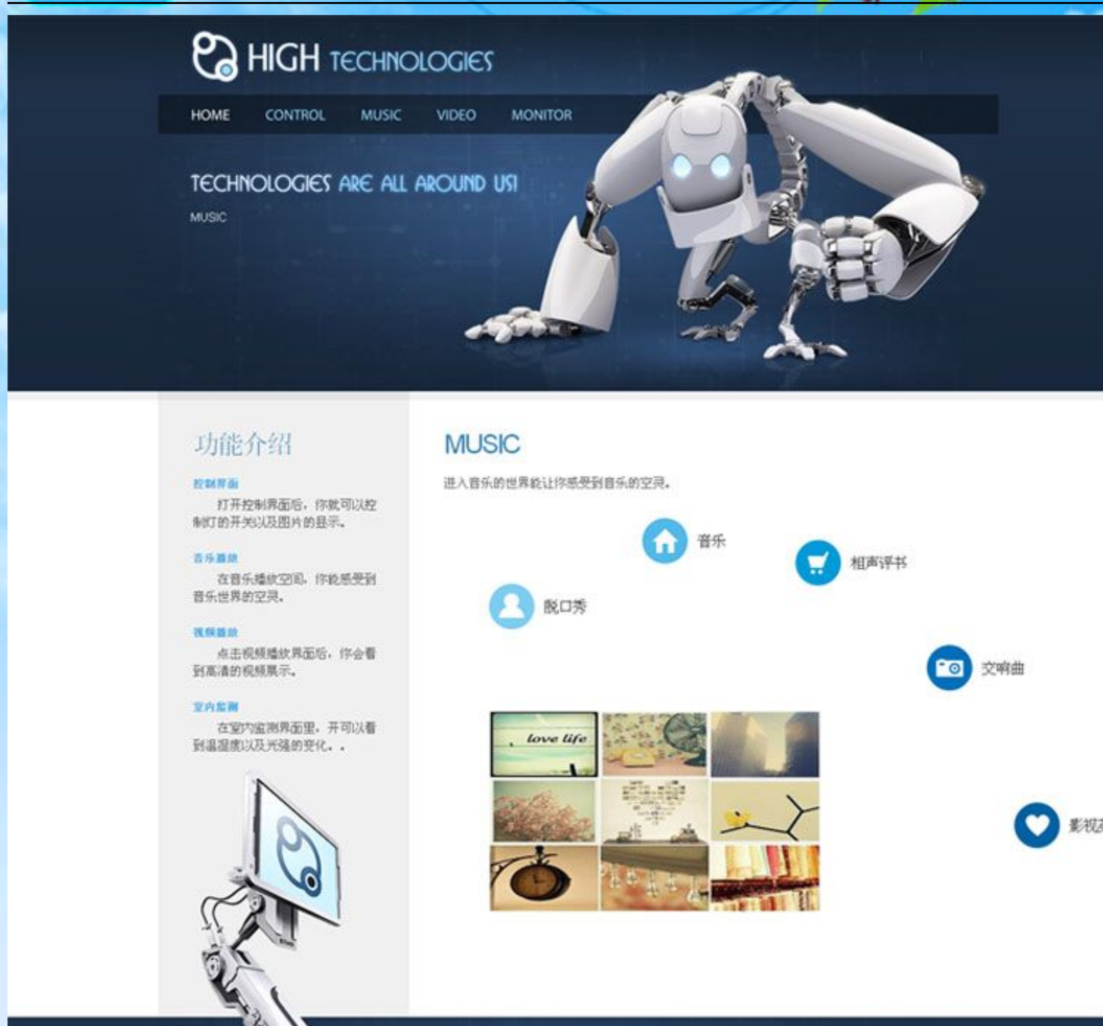


等待第一张图片显示完全，点击显示图片 2，显示图片 3，可以看到第二张和第三张图片：





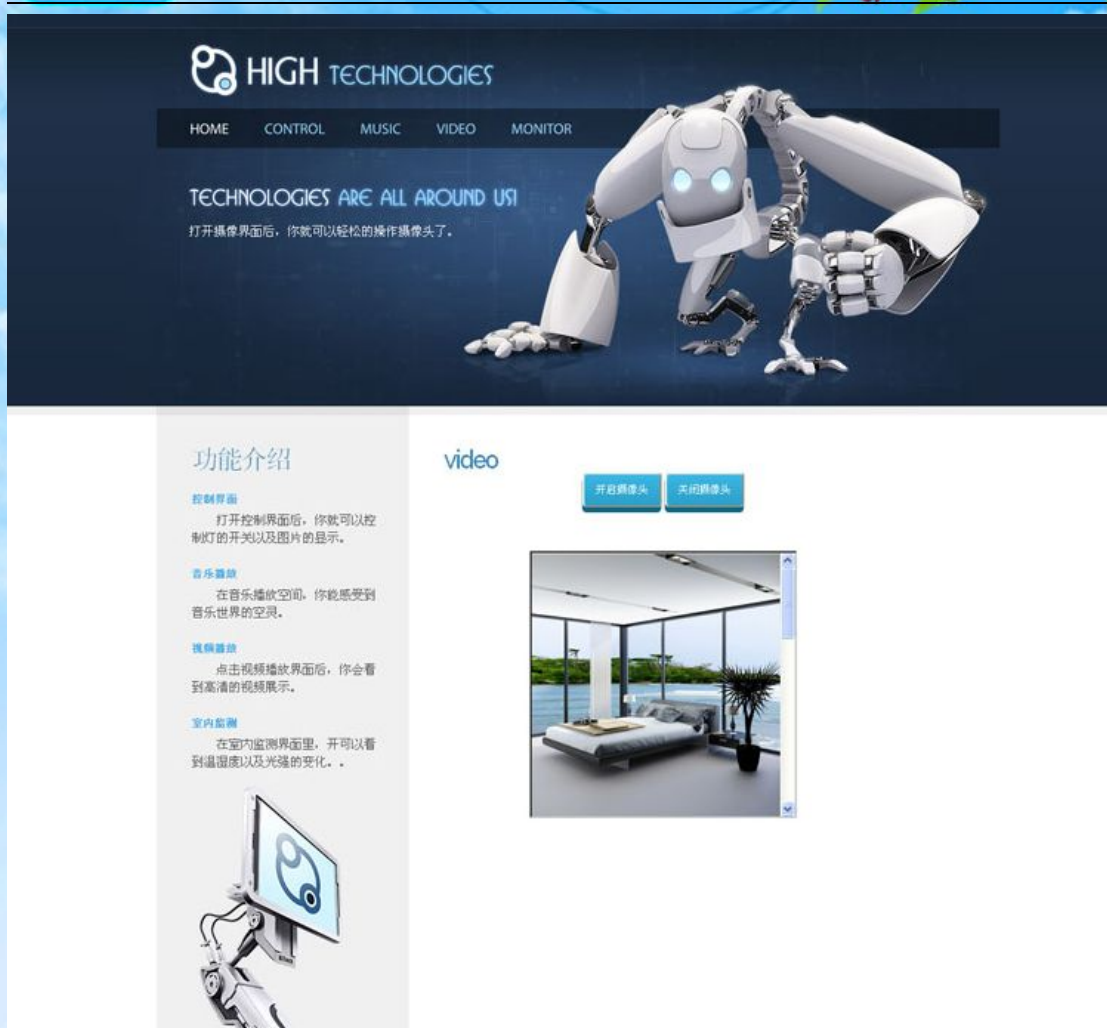
点击 MUSIC 可以看到很多的音乐播放的按钮，



光标移到脱口秀或者音乐的按钮上，点击相应的按钮可以听到不同的音乐，相声，脱口秀，影视英文等。



光标移到 VIDEO，可以跳转到摄像头监视界面：



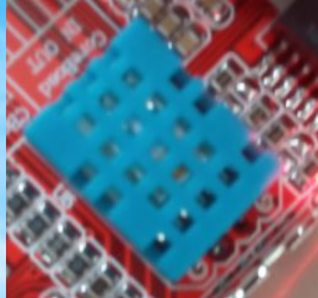
点击开启摄像头，动一动摄像头，就可以看到监视画面了。
点击 MONITOR，可以看到监测的当前的温度，湿度和光强强度：



用手捂住板子的光敏电阻，可以看到光强的值变小：



同样，在湿敏电阻附近升高温度（用打火机靠近）或湿度（加湿器），也可以看到温度和湿度的变化，湿度传感器的位置如下：



2.2局域网内 wifi 手机 app 控制

依然像之前一样连接开发板，给开发板上电，准备一部手机，安装微云电子提供的安卓 APP 软件：god.apk

安装完图标如下左边的图所示：



打开手机的 wifi，搜寻到名为 microcloud 的 ssid，密码为 12345678

打开界面如上图右边所示。在这个界面同样可以控制开发板的 led 灯的亮灭，和图片的切换。





在音乐播放一栏，可以控制每个音乐的播放与停止。
在监测界面可以看到温度湿度和光强的数值：

2.3 局域网内文件共享服务和网络流媒体

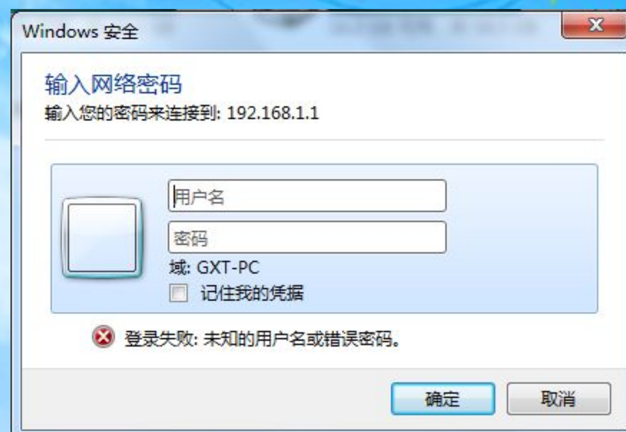
开发板上电之后，手机或者电脑连接上 microcloud 的无线 SSID 之后，可通过 Samba 共享服务在局域网内部查看和共享开发板上的文件，同时播放流媒体文件。

2.3.1　网络共享文件和流媒体视频——PC 机体验

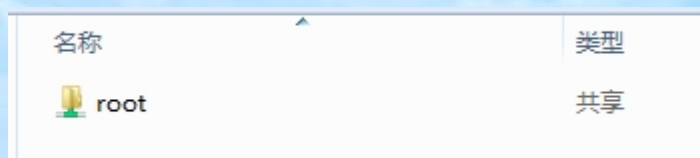
电脑连接 microcloud 的无线 SSID，打开“计算机”（win7 系统）或者“我的电脑”（XP 系统），在地址栏输入\\192.168.1.1，



点击回车，弹出如下界面：



输入用户名为 root，密码为 1，即可查看到开发板的文件夹：



双击打开，进入到 mnt/miclocloud/下，可以看到有很多的视频音频文件，随意打开一个音频文件，可以听到优美的音乐，



找到 microcloud.mp4，双击打开，可以通过本地的视频播放器播放：





文件夹内还有精美的电影（疯狂原始人 The_croods.mkv）：



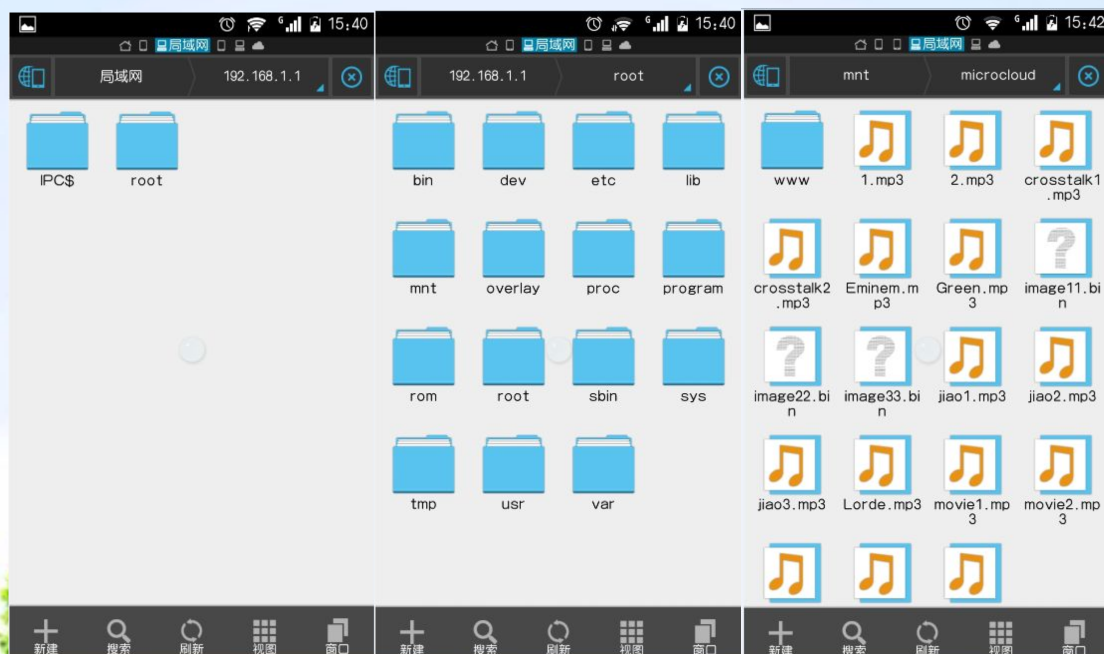


2.3.2 网络共享文件和流媒体视频——安卓手机体验

首先手机要下载一个 ES 文件浏览器，以便可以查看文件，具体如何下载这个软件不细说，下载完毕，打开如下图左图所示：

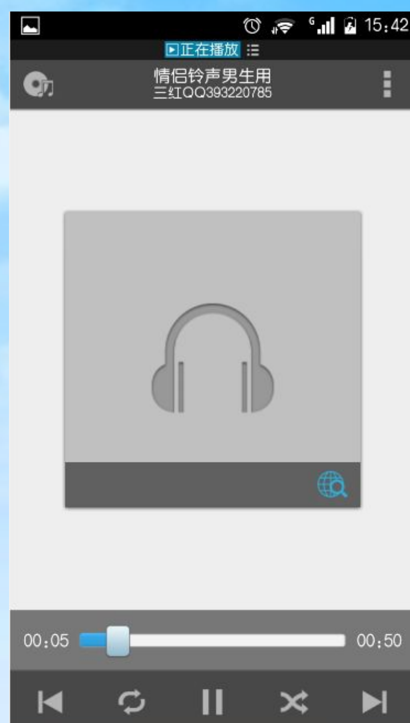


点击左上角的图标，选择局域网，如上面中间图所示，点击右下角的新建，输入上图右边图的的信息，创建开发板的服务器，创建好打开如下图左图所示，打开 root 文件夹，出现下图中间图画面，打开 mnt/microcloud/，出现下图右侧的画面：



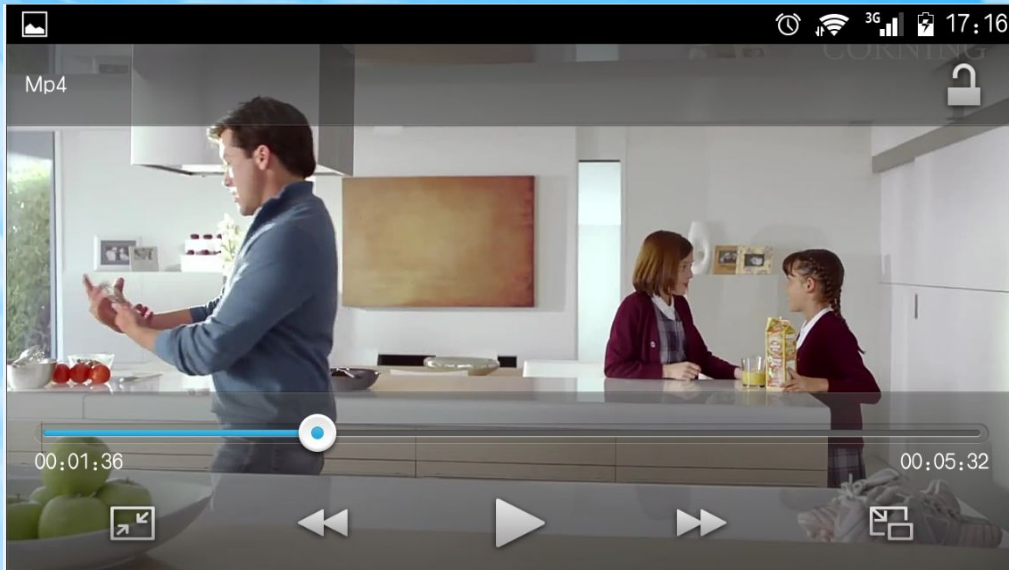


可以看到和电脑一样有很多的音频视频文件，随意打开一个 mp3格式的音频文件，选择手机上安装的音乐播放器，就可以听到优美的歌曲了：



双击打开 microcloud.mp4或者是 The_croods.mkv，可以通过本地的视频浏览器，观看视频，以下是截图：

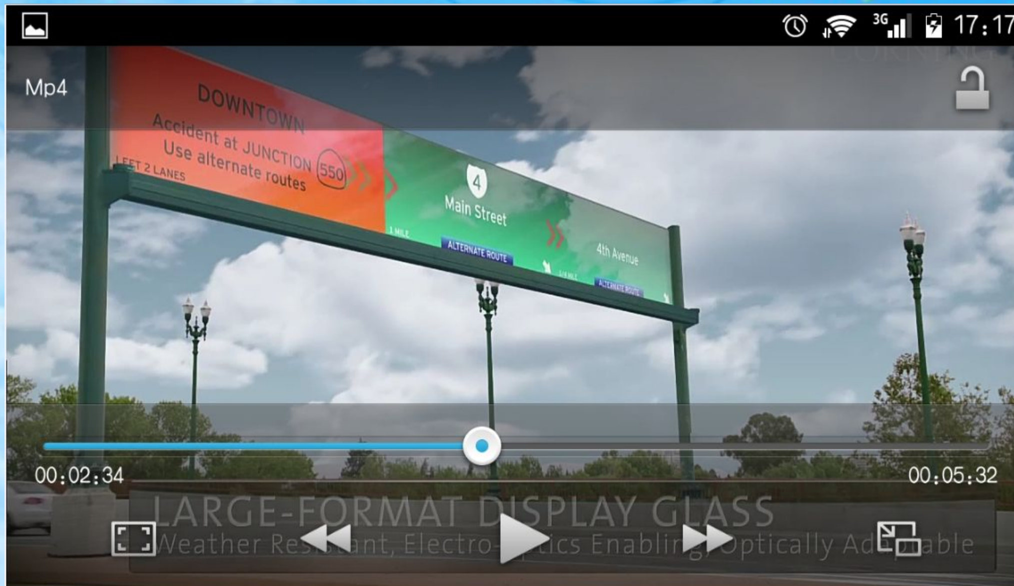




17:16
Mp4
00:01:36
00:05:32



17:17
Mp4
A Day Made of Glass
00:00:07
00:05:32



2.4Internet 远程智能控制

将开发板的 WAN 口连接上一个接入 Internet 的路由器上，可以使开发板接入外网，电脑通过无线连接上开发板，登陆 SSH 界面，键入如下命令，以开启外网控制功能：

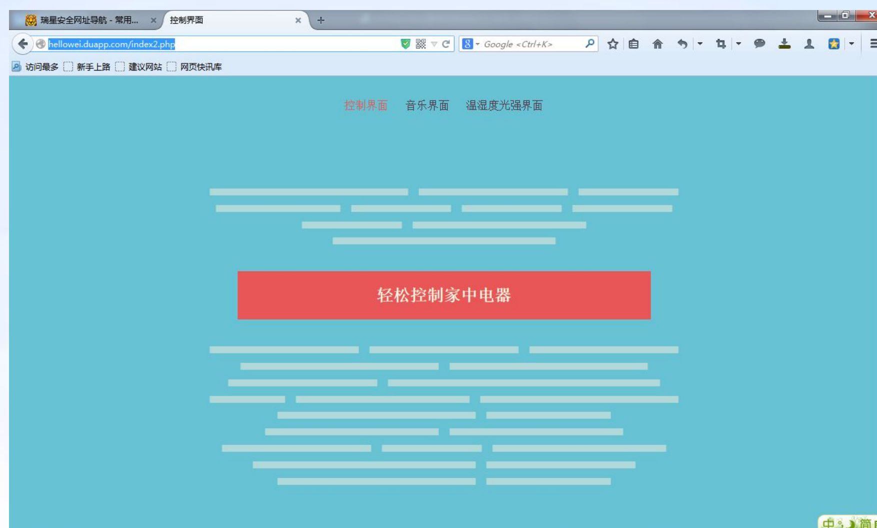
lua /mnt/microcloud/www/cgi-bin/hello.lua

```
root@MicroCloud:~# lua /mnt/microcloud/www/cgi-bin/hello.lua
```

在任意一台接入到 Internet 网络的主机上的火狐或谷歌浏览器里键入：

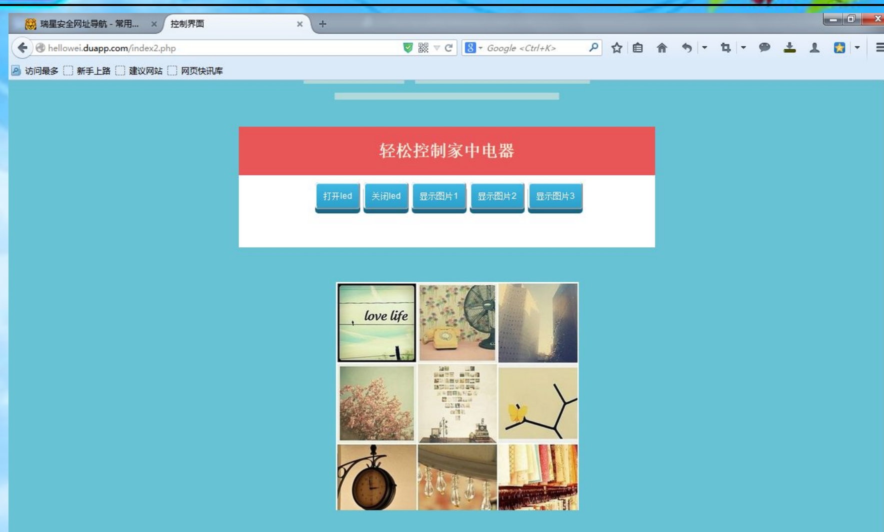
http://hellowei.duapp.com/index2.php

出现如下界面：

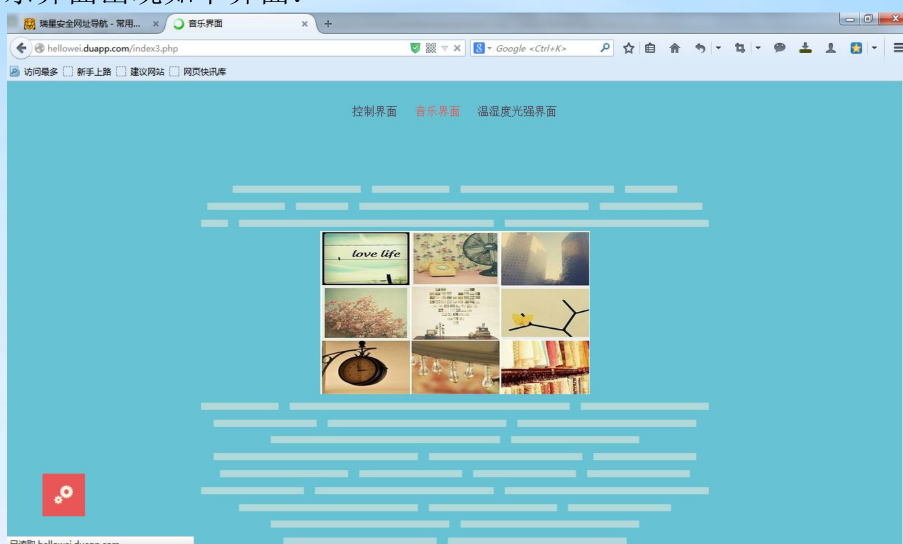


点击中间的红色区域，出现控制按钮，可以看到和局域网内一样的控制界面：

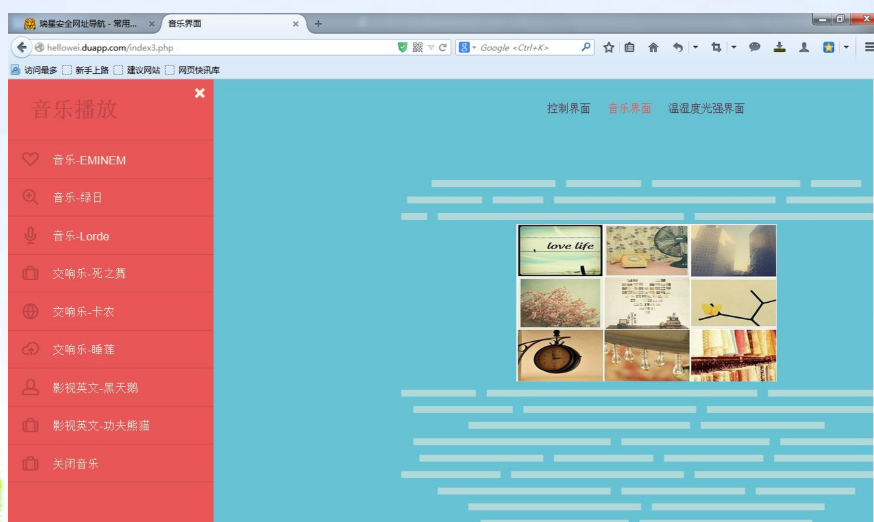




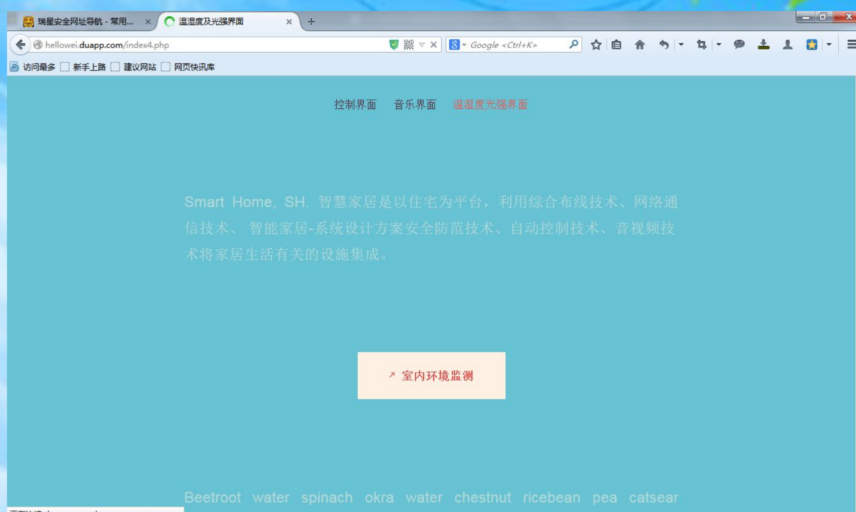
点击上图中的按钮同样可以控制开发板的 led 灯和图片的切换。
点击音乐界面出现如下界面：



点击左下角的红色按钮跳出音乐控制界面：



在这里可以选择各种音乐的播放，同时下面还有关闭音乐的按钮，可以随时随心的暂停音乐的播放。



点击中间的按钮可以显示当前的温度，湿度和光强。



第 3 章 STM32+LINUX 系统硬件资源及其电路

3.1 嵌入式硬件平台外围设备及其结构

我们的 STM32+LINUX 开发板的外观如图 2.1- 1 所示：

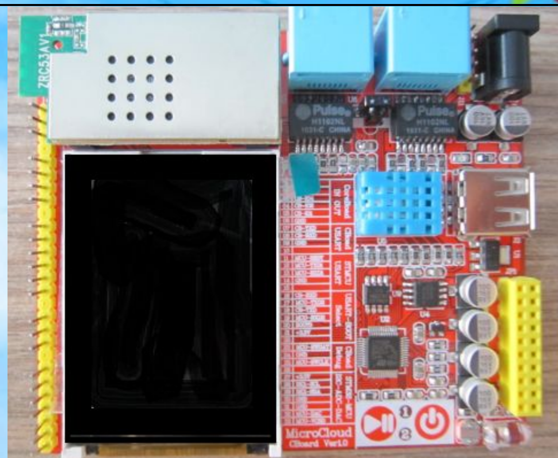


图 2.1- 1STM32+LINUX 开发板外观

STM32+LINUX 开发板的左上角最重要的模块：STM32+LINUX 开发板中央模块。



图 2.1- 2 中央模块

开发板正面的硬件架构如图 2.1- 3 所示：

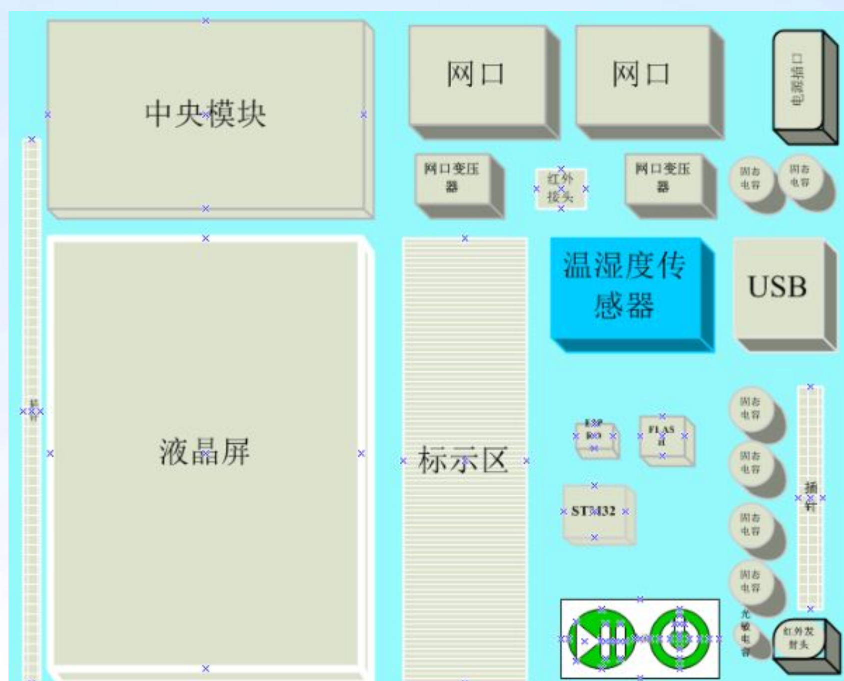


图 2.1- 3 基本硬件架构图

由上面两个图我们可以看到，开发板资源包括板载资源和附加资源，其中板载资源包括：

- 1 片中央模块，可嵌入 linux 系统。
- 2 组网口：与中央模块直接相连，可外接网线。
- 1 个电源接口，需 5V 适配器提供电源。
- 1 个 USB 接口，可外接摄像头，U 盘等设备。
- 1 个红外发射管，可实现红外遥控。
- 1 片 STM32 单片机，用于控制液晶屏、温湿度传感器、触摸按键等。
- 1 个 320*240 的彩色液晶屏，可显示清晰的图片。
- 1 个 Flash 闪存存储器，可存放图片信息。
- 1 个温湿度传感器，可测当前的温湿度。
- 2 个触摸式按键，由 STM32 控制。
- 1 个光敏电阻，可用于 AD 采集的测试。
- 1 片 EEPROM。

附加资源包括：

- 1 个免驱摄像头。
- 1 个可由红外控制的电源插排。
- 1 个 3G 网卡卡槽。
- 一个声卡。
- 一个 USB 扩展 HUB。

3.2硬件电路原理简介

3.2.1 一体化 Linux 模块及其原理

Linux 模块也就是 linux 嵌入式系统的处理模块,其的硬件原理图如图 2.2- 1 所示：

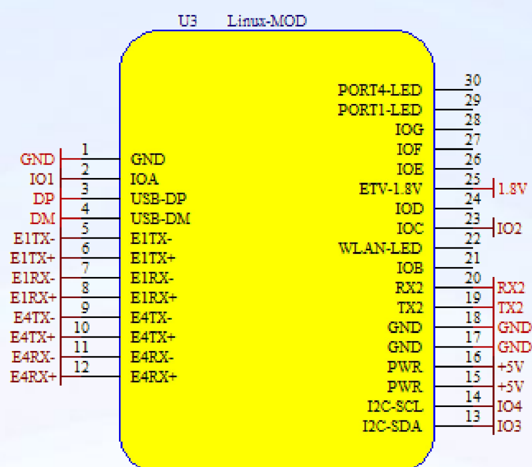


图 2.2- 1 中央模块

Linux 模块接出来的引脚有 USB 引脚（DM、DP、5V 和 GND），两组网口（八个引脚：E1TX+、E1TX-、E1RX+、E1RX-、E4TX+、E4TX-、E4RX+、E4RX-），一个串口（TXD、RXD），两个 IO 口（IO1 和 IO2）和一个 I2C（I2C-SCL 和 I2C-SDA）。这里我们常用到的是 USB 口，串口和网口，USB 可以用来插 U 盘，

摄像头，声卡等；串口用来显示开机启动信息、作为 linux 系统的终端、和 STM 32 进行通信等； 网口用来和电脑进行有线连接，实现网络的正常通信。
网口的电路原理图如图 2.2- 2 所示：

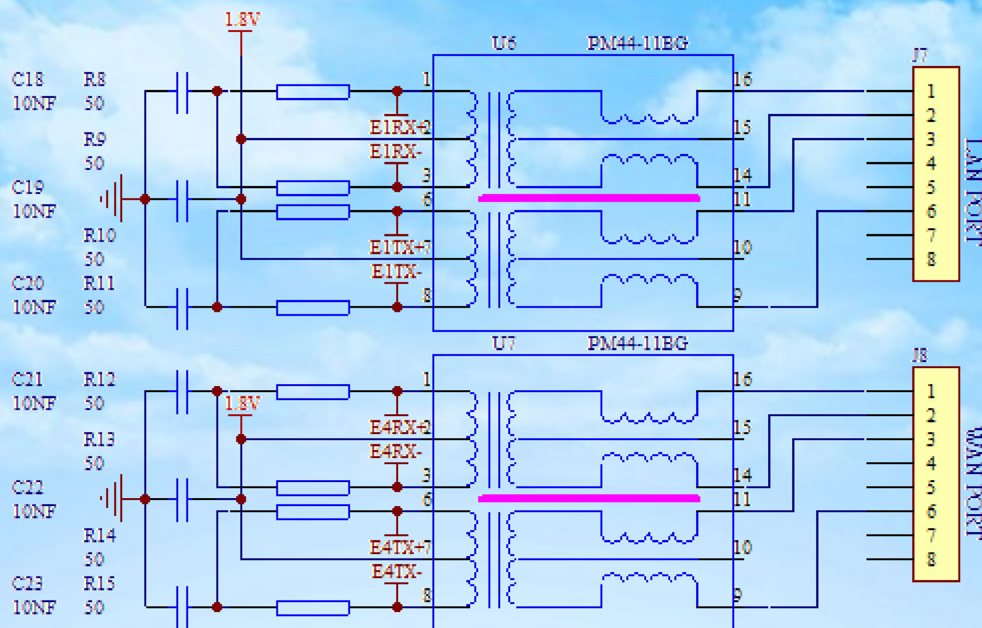


图 2.2- 2 网口的电路原理图

串口也已经被接出来了，如图 2.2- 3 所示:

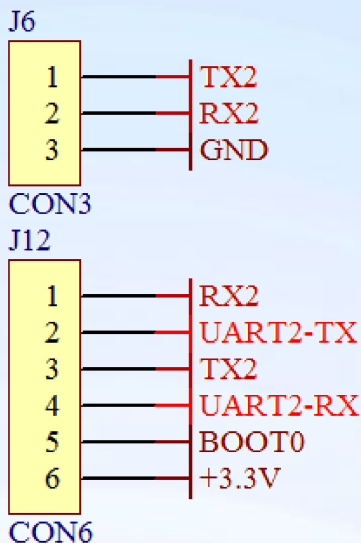


图 2.2- 3 串口的引脚原理图

串口引脚被引出来两次，一个用于做终端查看开机启动信息和输入 linux 命令，另一个可有跳帽控制，和 STM32 进行通信。USB 接口的电路如图 2.2- 4 所示：

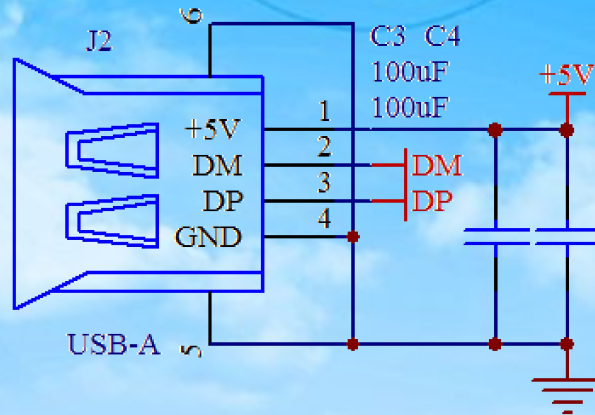


图 2.2- 4USB 硬件原理图

3.2.2 STM32 单片机及其原理

STM32 单片机的硬件原理图如图 2.2- 5 所示：

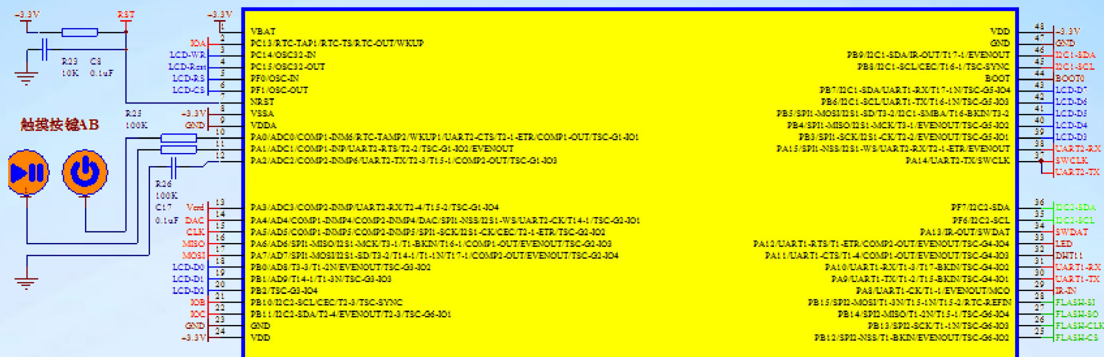


图 2.2- 5STM32 单片机硬件原理图

由上面的图上的引脚可以看出，STM32 所控制的外围器件有液晶屏、FLASH、EEPROM、触摸按键、SPI、串口、温湿度传感器、LED、光敏电阻、ADC 和 DAC。触摸按键的原理图在图 2.2- 5 上显示了，由两个按键和一个采集电容组成

1）液晶屏的原理图如图 2.2- 6 所示：

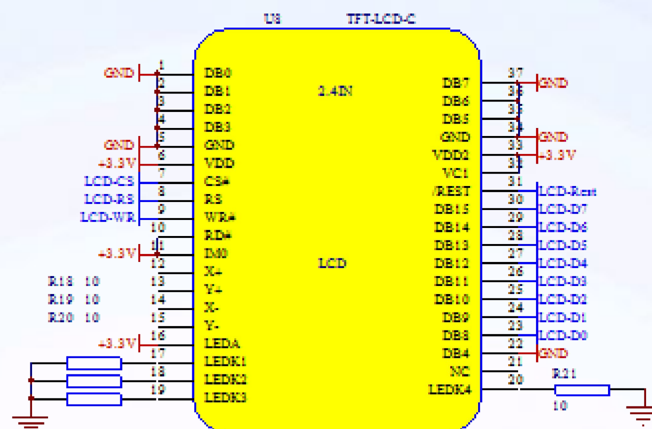


图 2.2- 6 液晶屏原理图

液晶屏和 STM32 的硬件接口有 8 位并行数据口 LCD_Dx，片选口 LCD_CS，锁存数据引脚 LCD_WR，写命令或者是数据引脚 LCD_RS 和复位引脚 LCD_Rest。

2）FLASH 闪存存储器由 STM32 的 SPI2 总线控制，需要控制的引脚包括片选 CS，数据主发从收 MOSI，数据主收从发 MISO，和时序 SCK，如图 2.2- 7 所示：

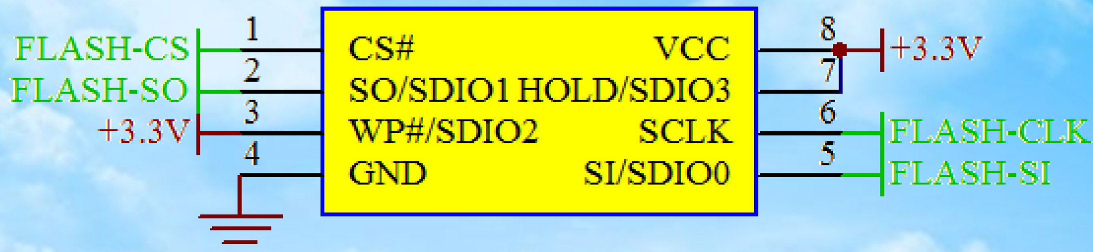


图 2.2- 7FLASH 原理图

3）温湿度传感器由普通的 GPIO 口操作，属于一线总线，原理图如图 2.2- 8 所示：

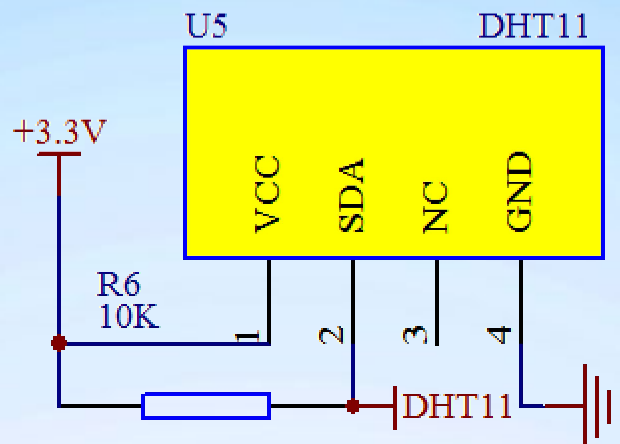


图 2.2- 8 温湿度传感器原理图

4）LED 和红外发射的操作也是 IO 口操作，光敏电阻是 AD 采集，原理图如图 2.2-9 所示：

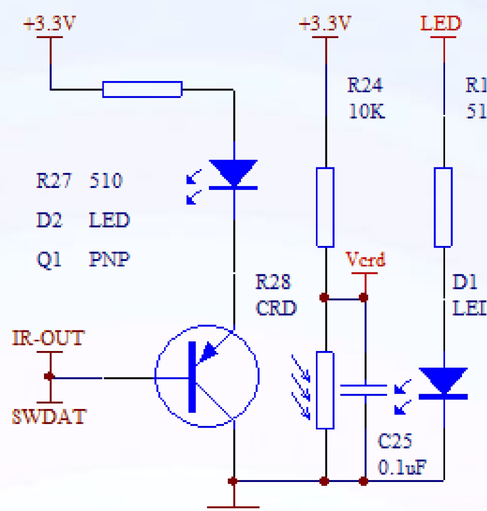


图 2.2- 9LED、光敏电阻和红外发射的原理图

至此，本教程的实验平台（微云 STM32+LINUX 开发板）的硬件部分就介绍完了，了解了整个硬件对我们后面的学习会有很大帮助，有助于理解后面的代码，在编写软件的时候，可以事半功倍。

STM32 单片机详解篇

第 4 章 RVMDK 软件入门

4.1 MDK4.70a 的安装与破解

4.1.1 MDK4.70a 安装步骤

我们以为 KEIL 公司的 RVMDK4.70a 的 STM32 开发环境为例，介绍一下它的安装与破解。打开资料包里的软件资料文件夹，将名字为 MDK470 的压缩包解压，得到以下两个文件：

keygen	2008/2/27 17:42	应用程序	18 KB
mdk470a	2014/2/20 8:54	应用程序	521,720 KB

这是 MDK 的安装文件，和安装其他软件一样，相信大家都会明白，一直点击 Next 直到出现下面界面后，随意填写好您的信息，这些信息其实没什么要求，可以随意填写，然后点击 Next。

Setup MDK-ARM V4.70a

Customer Information

Please enter your information.

KEIL Tools by ARM

Please enter your name, the name of the company for whom you work and your E-mail address.

First Name: tian

Last Name: tianzi

Company Name: microcloud

E-mail: xxxxxxxxx@163.com

Keil MDK-ARM Setup

<< Back | Next >> | Cancel

图 1- 1 填写个人信息

接着出现下面图 1- 2 的界面，第二个选项是在打开软件的时候添加一个默认的工程，这个不重要，可以不用理会。

Setup MDK-ARM V4.70a

File installation completed

KEIL Tools by ARM

µVision Setup has installed all files successfully.

Retain current µVision configuration.

Add example projects to the recently used project list.

Preselect Example Projects for

Keil MDK-ARM Setup

<< Back | Next >> | Cancel

图 1- 2 默认

按图 1- 3 中所示选择之后点击“Finish”之后，MDK 便安装完成。



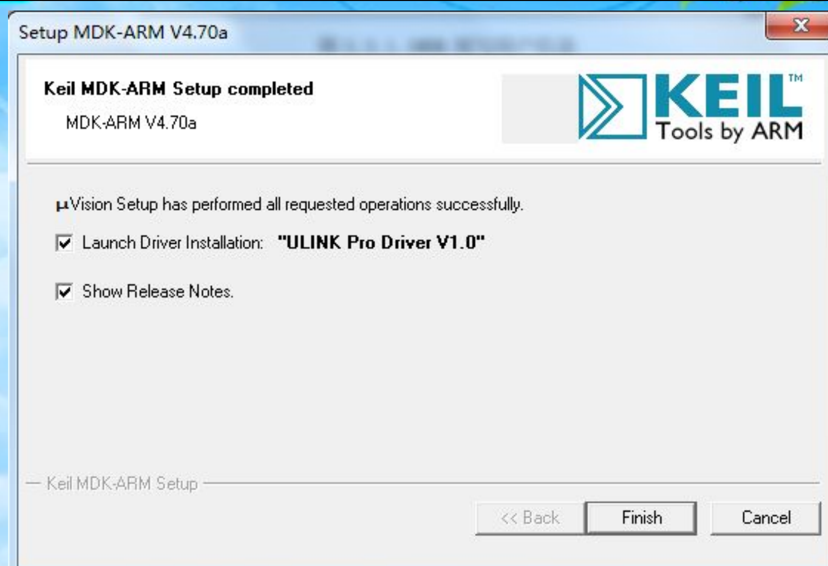


图 1- 3 安装完成

4.1.2 添加 License Key

MDK 针对每台机会有一个 CID，copy 这个 CID 到注册机处生成 License Key，然后再将这个 License Key 添加到 MDK 里面去注册。

打开运行 MDK。这里要注意，有些版本的 windows 系统(如 Vista)需要右键点击快捷方式选择“以管理员身份运行”，因为注册 license 需要管理员权限。打开 MDK 后有一个名字叫“LPC2129 simulator”的默认 Project，暂时我们可以不用理会。

如图 1- 4 点击：File->License Management,弹出一个 License Management 界面（如图 1- 5）,copy 界面中的(CID)：

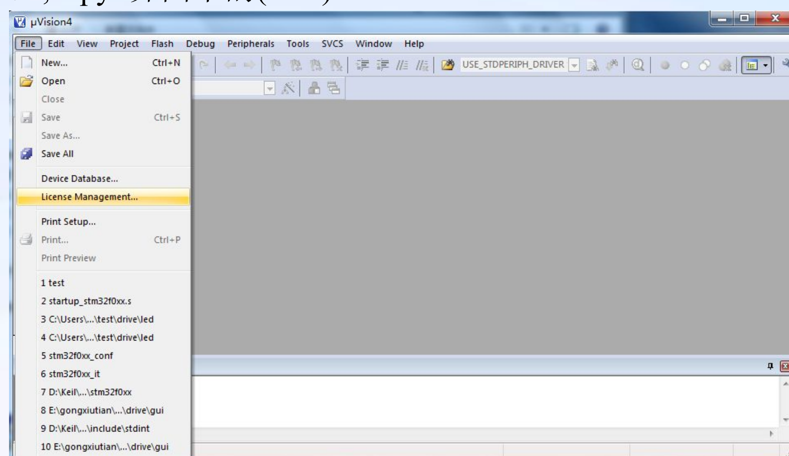


图 1- 4 进入 License Management



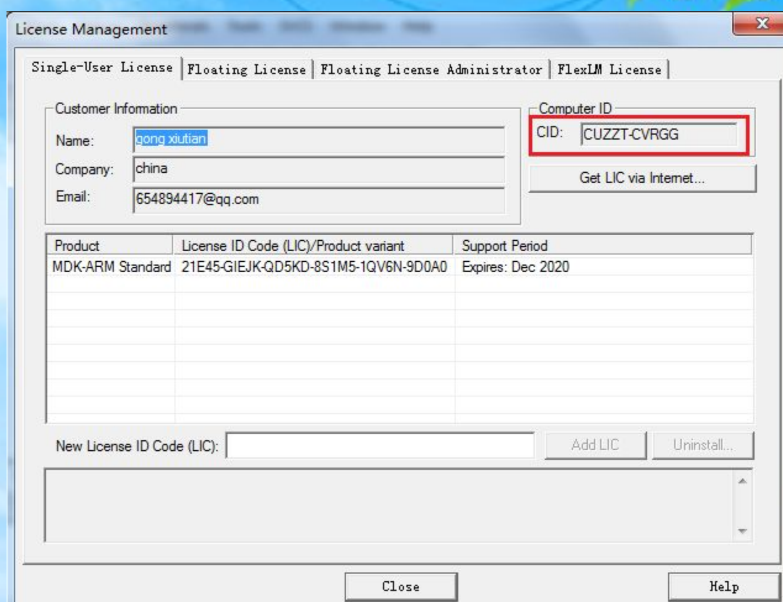


图 1- 5 copy 界面中的(CID)

打开资料包（软件资料\MDK4.70a\keygen.exe)下面的注册机，注册机我们会跟 MDK 安装包放在同一目录下面。最好以管理员的身份运行软件。

接着会出现注册界面如图 1- 6，黏贴刚才复制的 CID 到 CID 输入框，然后 Target 选择 ARM 之后，点击“Generate”，30 位的 License Key 会在下图红色标出的部分生成。License Key 的格式：D0DY8-30KAK-0N8AM-X9Z14-A2NWP-J3LZZ。

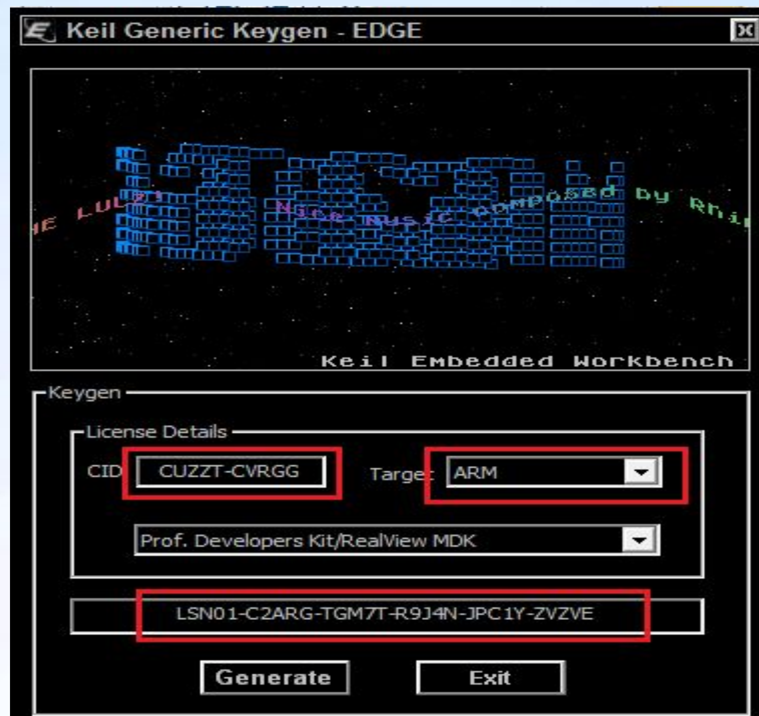


图 1- 6 注册机

如图 1- 7 将这个 License Key 黏贴到 Keil 的 License Management 界面的 New License Id Code 一栏，然后点击“Add LIC”，添加成功后会出现成功提示。然后点击 Close 关闭这个界面即可。到此 License Key 便添加完成。

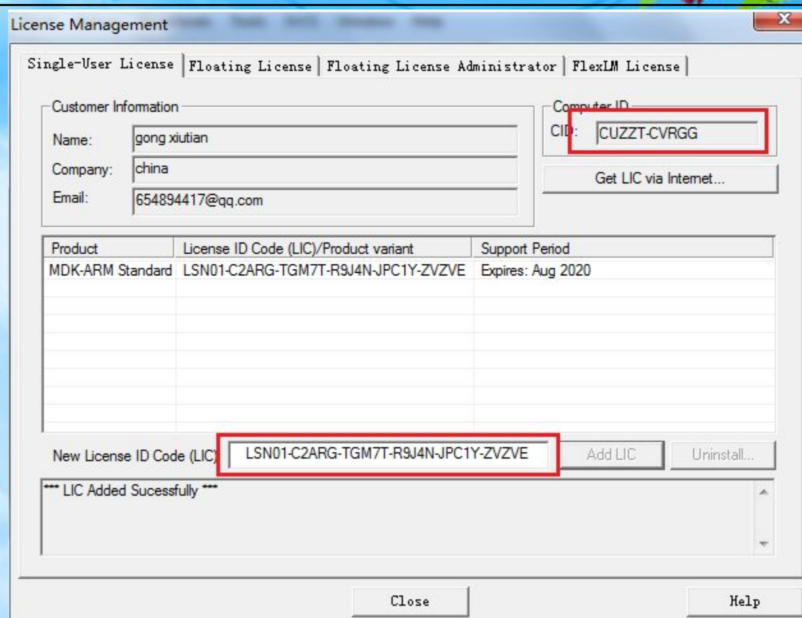


图 1- 7 破解成功

4.2新建工程模板

在建立工程之前，我们建议用户在电脑的某个目录下面建立一个文件夹，后面所建立的工程都可以放在这个文件夹下面，这里我们建立一个文件夹为 first_project。

如图 1- 8 点击 Keil 的菜单：Project －>New Uvision Project ，然后将目录定位到刚才建立的文件夹 first_project 之下，在这个目录下面建立子文件夹 user(我们的代码工程文件都是放在 user 目录，很多人喜欢新建“Project”目录放在下面，这也是可以的，这个就看个人喜好了)，然后定位到 user 目录下面，我们的工程文件就都保存到 user 文件夹下面如图 1- 9。工程命名为 first_project，点击保存。

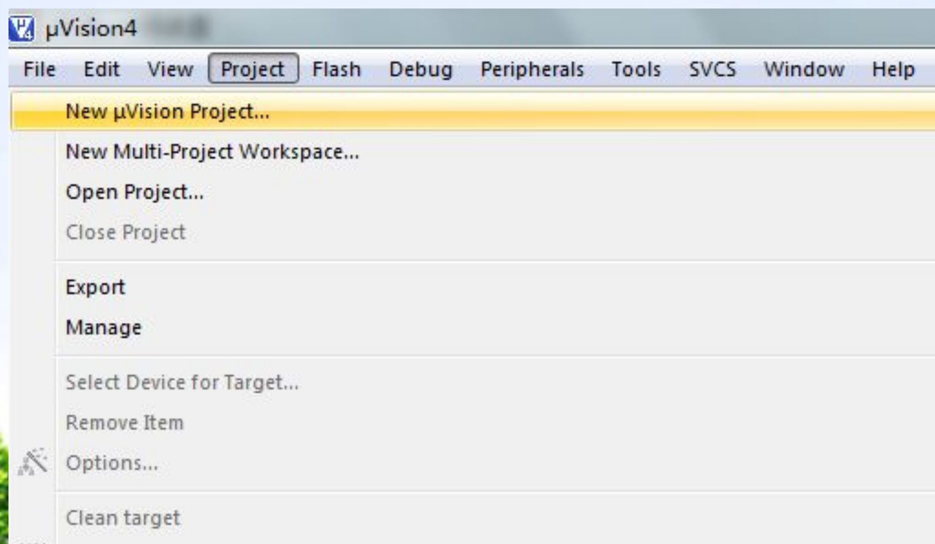


图 1- 8 新建工程

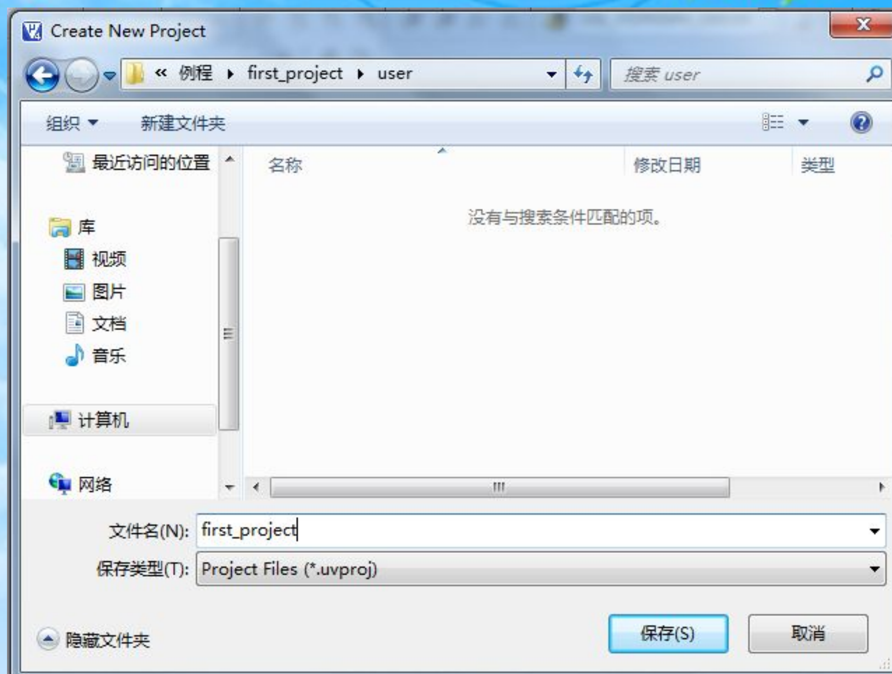


图 1- 9 保存工程名

接下来会出现一个选择 Device 的界面如图 1- 10，就是选择我们的芯片型号，这里我们定位到 STMicroelectronics 下面的 STM32F051R8(针对我们的开发板是这个型号，如果是其他芯片，请选择对应的型号即可)。

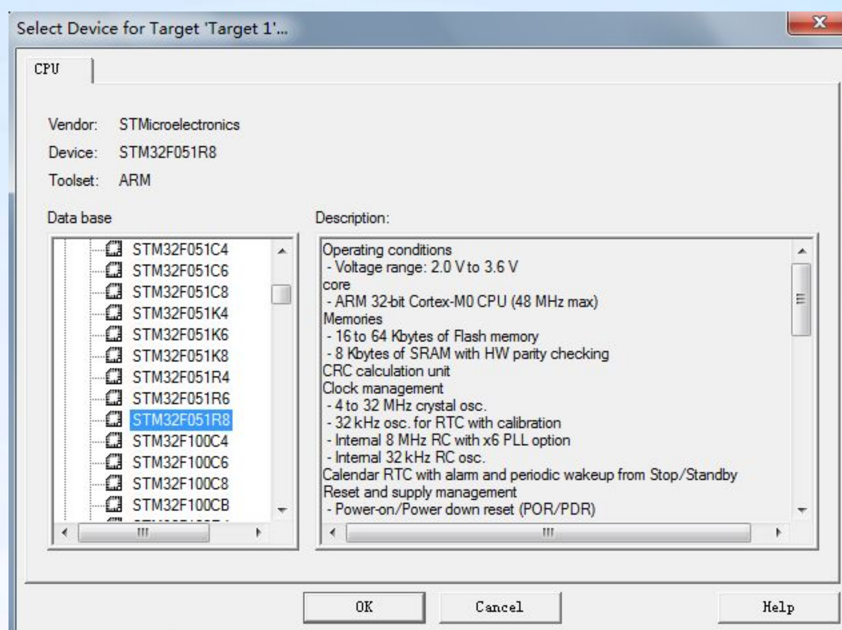


图 1- 10 选择芯片

弹出对话框“Copy STM32 Startup Code to project ….”，询问是否添加启动代码到我们的工程中，这里我们选择“是”。

现在我们看看 user 目录下面包含两个文件如图 1- 11：



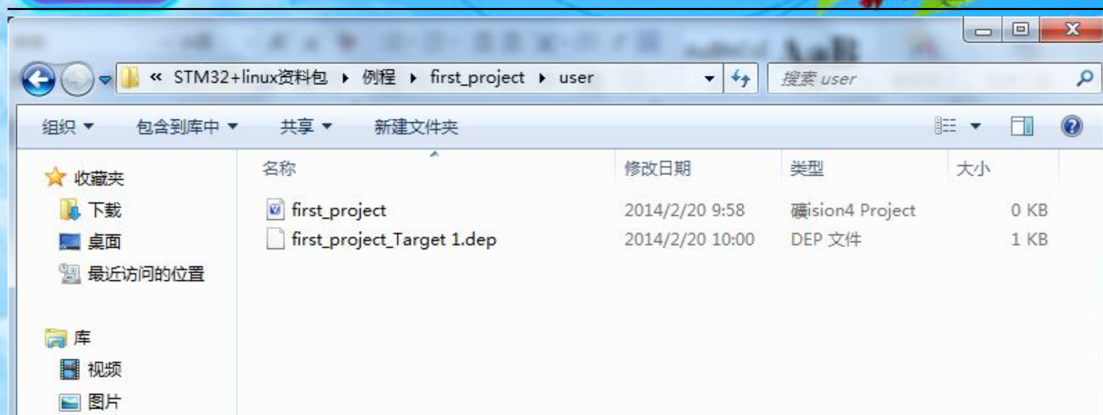


图 1- 11user 目录

接下来，如图 1- 12 我们在 first_project 工程目录下面，将开发板的任意工程目录下的 drive，stm32f0 和 lib 三个文件夹复制到本工程的目录下面，stm32f0 用来存放核心文件 stm32f0xx_conf，stm32f0xx_it，system_stm32f0xx，这些文件我们不需要修改，lib 文件夹顾名思义用来存放 ST 官方提供的库函数源码文件；drive 文件夹存放的是微云 STM32+linux 开发板的驱动文件，是我们学习的主要内容；已有的 user 目录除了用来放工程文件外，还用来存放主函数文件 main.c，以及其他链接文件。

名称	修改日期	类型	大小
drive	2014/2/20 10:17	文件夹	
lib	2014/2/20 10:17	文件夹	
stm32f0	2014/2/20 10:17	文件夹	
user	2014/2/20 10:00	文件夹	

图 1- 12 工程文件

如图 1- 13 右键点击 Target1，选择 Manage Components

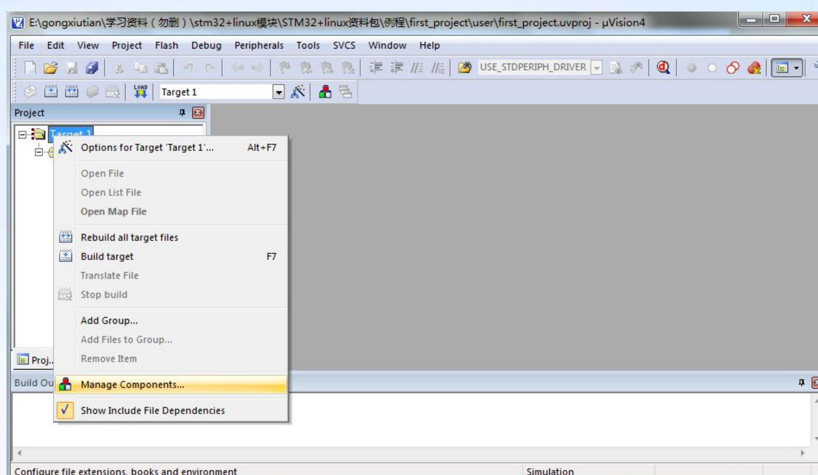


图 1- 13 右键点击 Target1

如图 1- 14，Project Targets 一栏，我们将 Target 名字修改为 first_project,然后建立四个 Groups：user,stm32f0,lib 和 drive。然后点击 OK，如图 1- 15 可以看到我们的 Target 名字以及 Groups 情况。

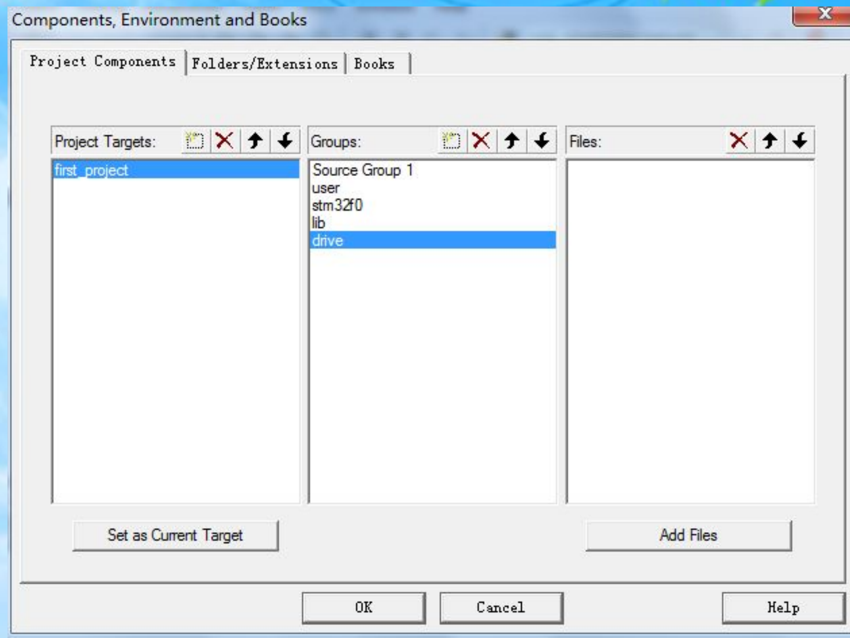


图 1- 14 Project Targets 界面

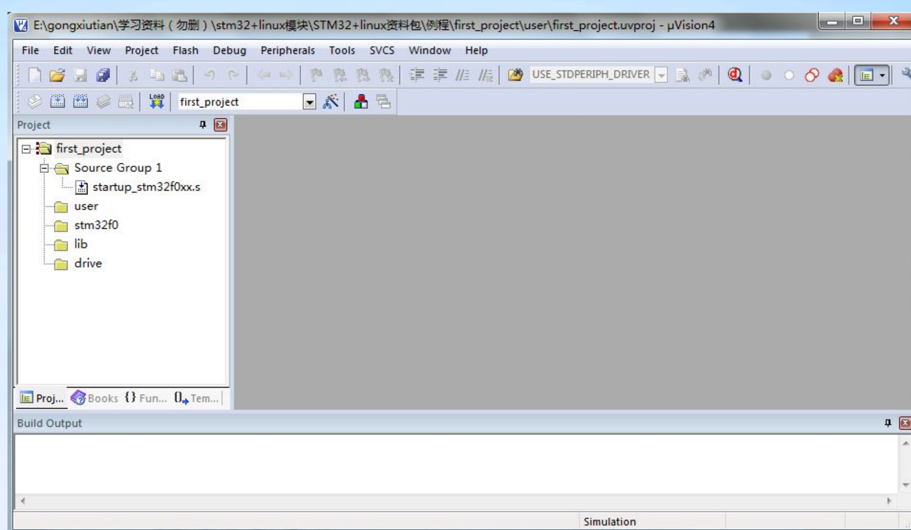


图 1- 15 Target 名字及 Groups 情况

下面我们往 Group 里面添加我们需要的文件。我们按照步骤 12 的方法，右键点击 first_project，选择选择 Manage Components.然后选择需要添加文件的 Group，如图 1- 16 这里第一步我们选择 lib ，然后点击右边的 Add Files, 定位到我们刚才建立的目录 lib/src 下面，将里面所有的文件选中 (Ctrl+A)，然后点击 Add，然后 Close.可以看到 Files 列表下面包含我们添加的文件。

这里需要说明一下，对于我们写代码，如果我们只用到了其中的某个外设，我们就可以不用添加没有用到的外设的库文件。例如我只用 GPIO，我可以只用添加 stm32f0xx_gpio.c 和 stm32f0xx_rcc.c 而其他的可以不用添加。这里我们全部添加进来是为了后面方便，不用每次添加，当然这样的坏处是工程太大，编译起来速度慢，用户可以自行选择。

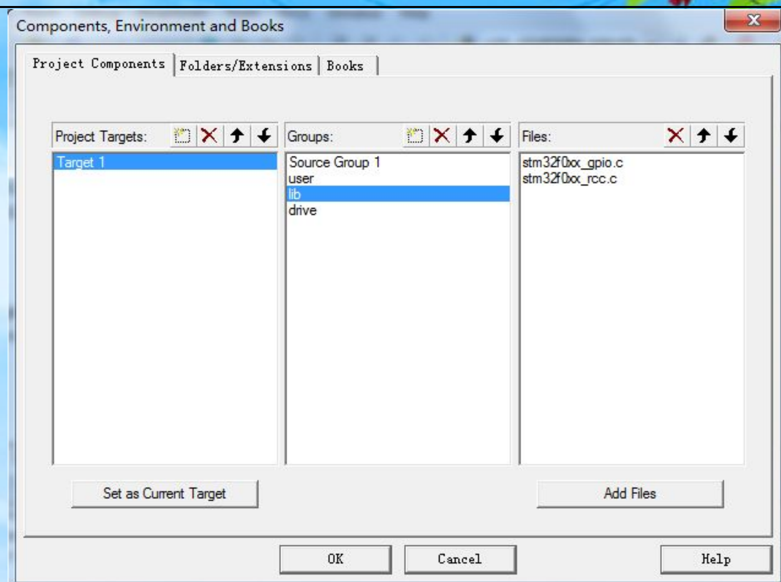


图 1- 16 选择 lib

用同样的方法，将 Groups 定位到 drive，stm32f0 和 user 下面，添加需要的文件。这里我们的 stm32f0 下面需要添加的文件为 stm32f0xx_it.c（中断会用到，如果工程不涉及中断可以不用添加），system_stm32f0xx.c。user 目录下面需要添加的文件为 main.c(这里我们添加的是“闪烁 led”工程目录下的主函数)，drive 里添加 led.c（以点亮 led 灯为例）。

这样我们需要添加的文件已经添加到我们的添加中去了，最后点击 OK，回到工程主界面。

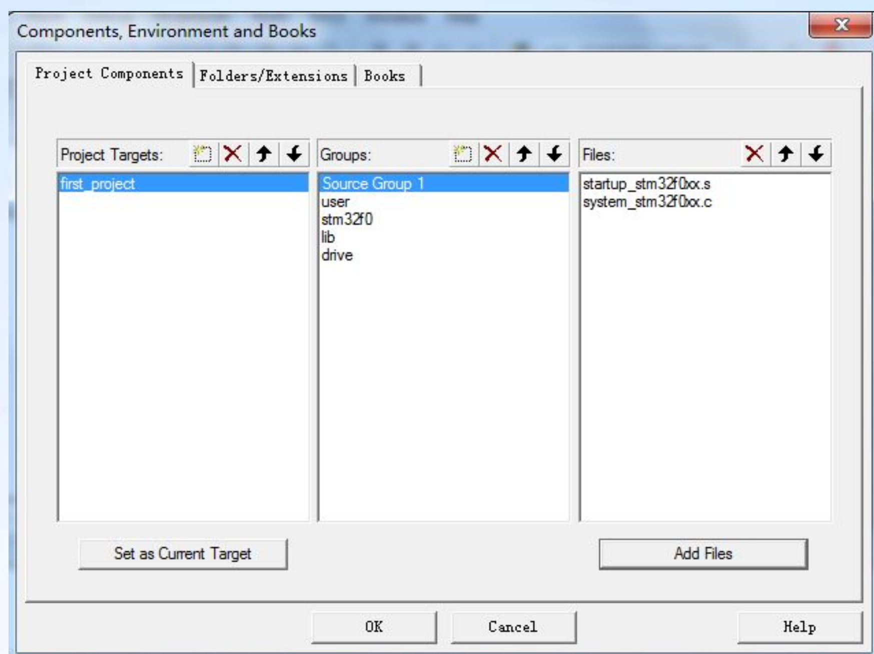


图 1- 17 添加文件

如图 1- 18，回到工程的主页面之后我们需要新建 main.c 文件，点击新建按钮，然后点击保存，将文件保存在工程的 user 文件夹下，回到工程界面，将光标移到右侧 user 组上，右击，选择 Add Files to Group ‘user’....将主函数 main.c 添加进来。以相同的方式将 led.c 添加的 Group drive 里。

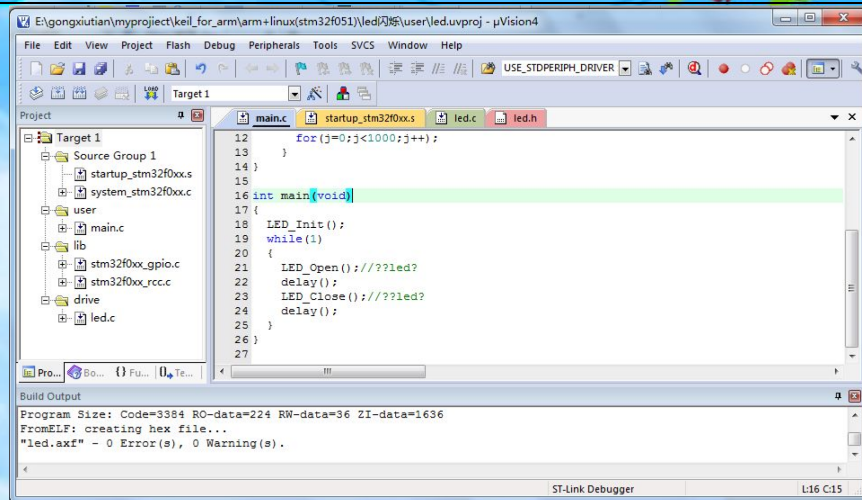


图 1- 18 新建 main.c

接下来，我们需要配置工程，否则编译会出错误。

回到工程主菜单，如图 1- 19 点击魔术棒，出来一个菜单如图 1- 20，然后点击 c/c++选项.在 define 一栏里添加 USE_STDPERIPH_DRIVER STM32F0XX，然后点击 Include Paths 右边的按钮。如图 1- 21 弹出一个添加 path 的对话框，然后我们将图上面的 3 个目录添加进去。记住，keil 只会在一级目录查找，所以如果你的目录下面还有子目录，记得 path 一定要定位到最后一级子目录。然后点击 OK.

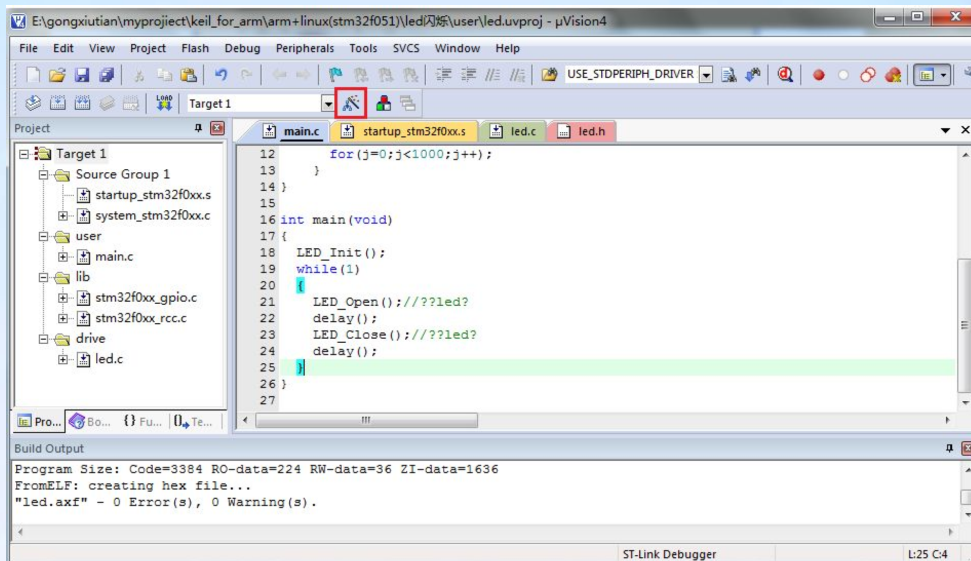


图 1- 19 点击魔术棒

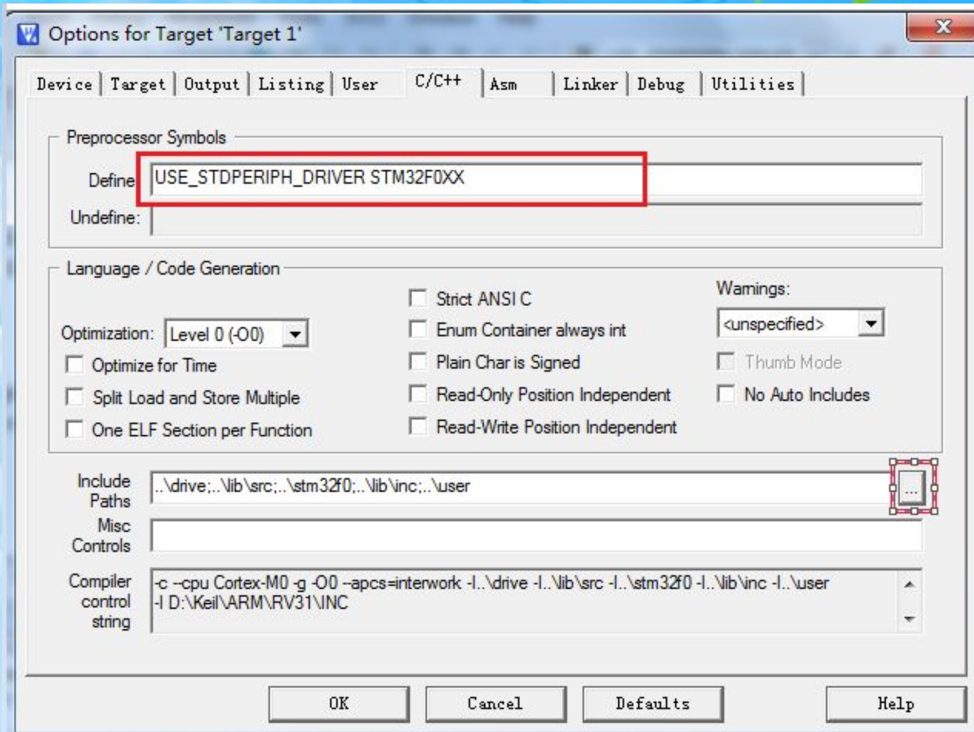


图 1- 20 魔术棒菜单

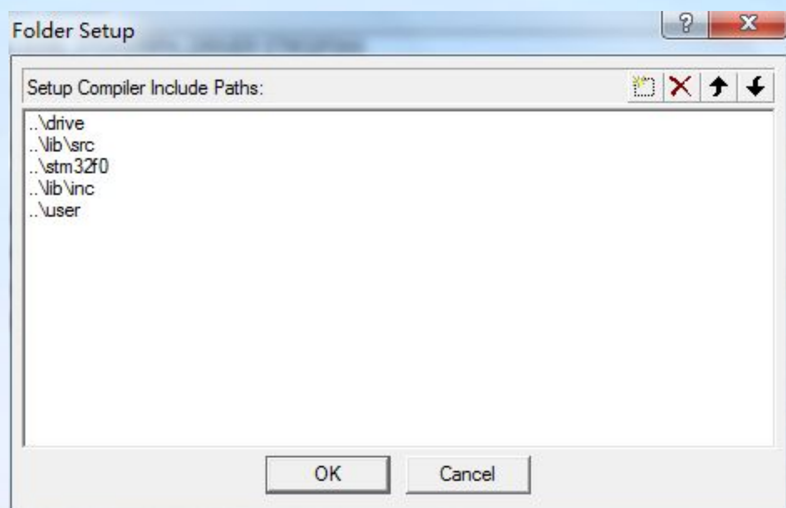


图 1- 21 添加 path

然后回到主页面，点击编译一下，应该就没有错误了。

这样一个工程模版建立完毕。下面还需要配置，让编译之后能够生成 hex 文件。同样点击魔术棒，进入配置菜单，选择 Output。如图 1- 22 然后勾上下三个选项。 其中 Create HEX file 是编译生成 hex 文件，Browser Information 是可以查看变量和函数定义。

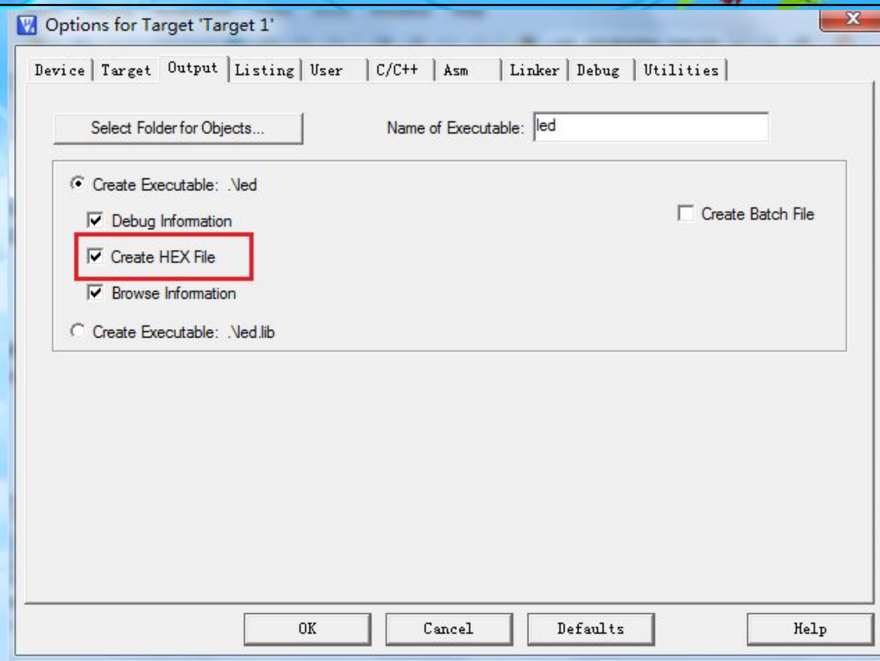


图 1- 22 Create HEX file

到这里，一个基于固件库的工程模板就建立了。

4.3CH340 驱动安装及 COM 查看

4.3.1 在 win7 系统下安装 CH340 驱动

在 win7 系统下安装 CH340 驱动是为了 USB 线连接电脑与单片机的时候，使电脑中可读出单片机连接的 COM 口。安装步骤如下。

1、 鼠标左键双击 CH340 文件夹，如图 1- 23 所示。

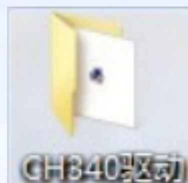


图 1- 23CH340 文件夹

2、 打开 CH340 文件夹后，如图 1- 24 所示。

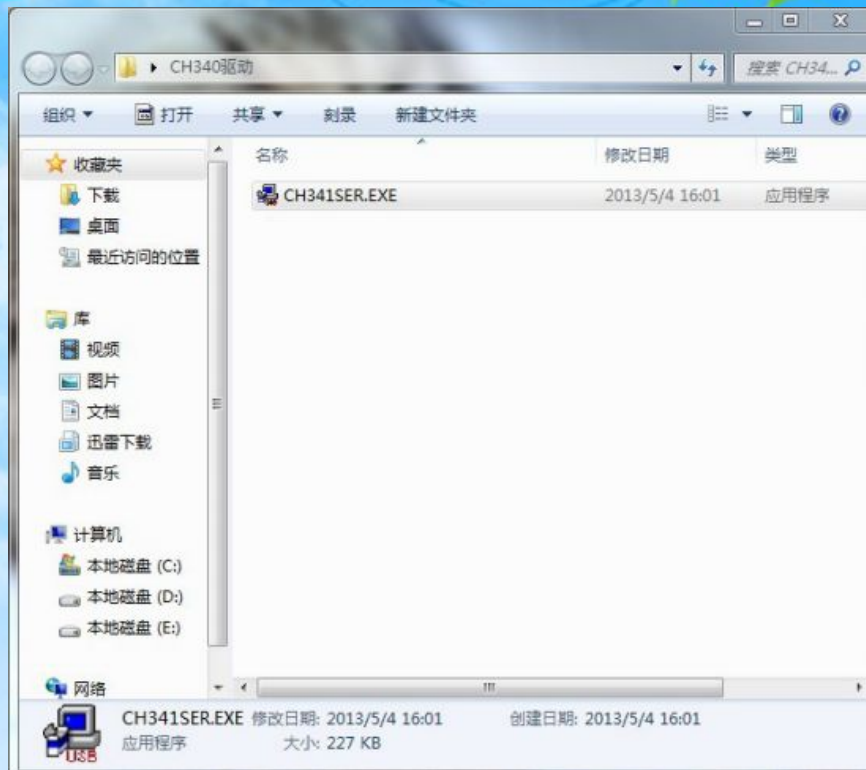


图 1- 24CH340 文件夹内容

3、鼠标右键单击 CH341SER.EXE，选择以管理员身份运行，如图 1- 25 所示。

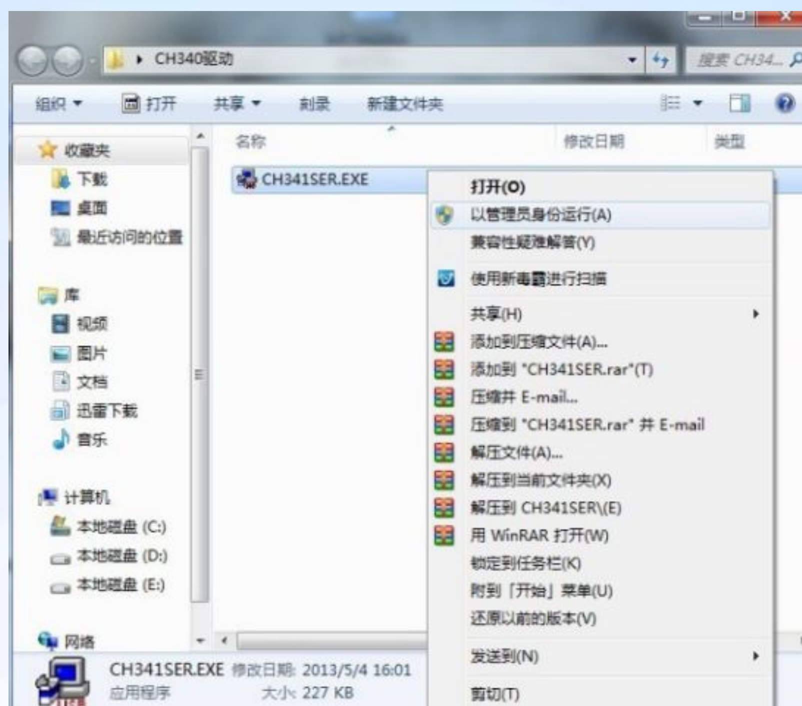


图 1- 25 管理员身份运行

4、运行后出现驱动安装窗口，如图 1- 26 所示。



图 1- 26 驱动安装窗口

6、单击“安装”，出现驱动安装成功，如图 1- 27 所示。



图 1- 27 安装成功

7、单击确定，关闭驱动安装即可。

4.3.2 如何在 win7 系统下查看 com 口

1、 将基础型学习板与计算机用 USB 线连接， 连接成功后， 右键单击计算机，

选择管理，如图 1- 28 所示。



图 1- 28 计算机界面

2、选择管理后界面如图 1- 29 示。

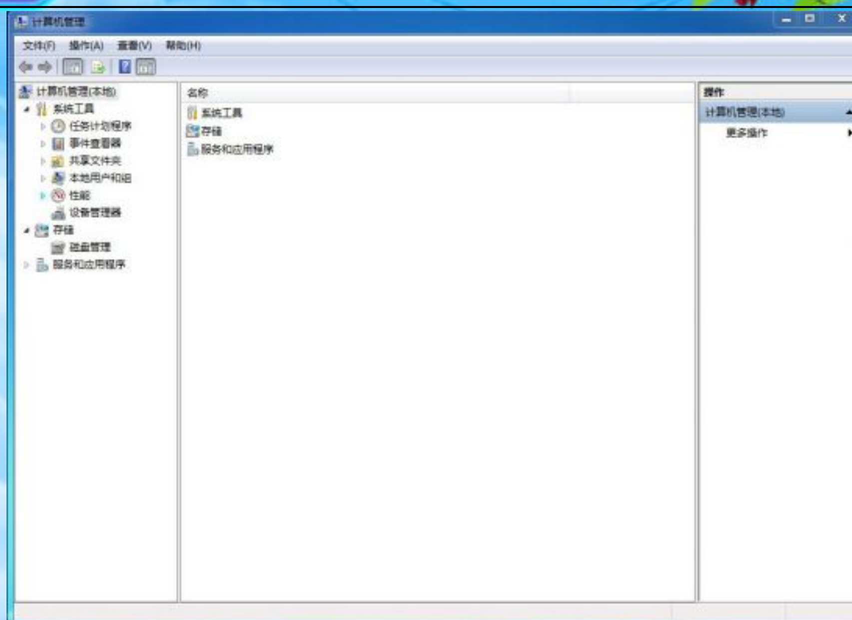


图 1- 29 单击管理后界面

3、 选择“计算机管理”界面左侧的**设备管理器**，如图 1- 30 所示。



图 1- 30 选择设备管理器

4、 在弹出的选项中选择“端口（COM 和 LPT） ” ，如图 1- 31 所示。

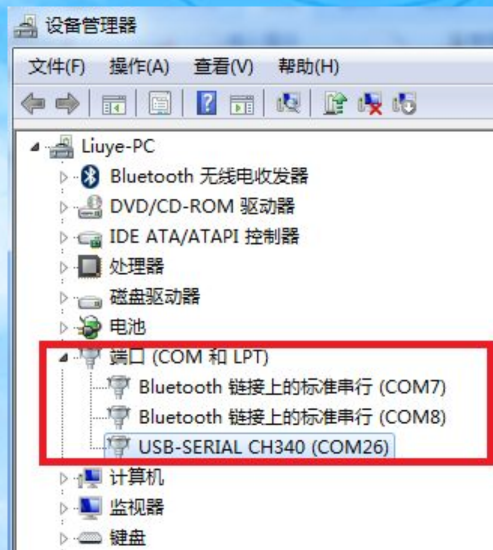


图 1-31 端口

5、 由图可知， 此 COM 口为 COM26， 读者的计算机 COM 口不一定是 COM26，以实际情况为准。

4.4MDK 下的程序下载和硬件调试

STM32 的程序下载有多种方法：USB、串口、JTAG、SWD 等，这几种方式，都可以用来给 STM32 下载代码。这里我们介绍一下 USB 下载和 ST-LINK 下载。

4.4.1 串口下载

要实现此功能我们需要：

1、安装两个软件。

①CH340 驱动的安装，请参照上一小节。

②安装 Flash_Loader_Demonstrator_v2.6.0_Setup 官方下载软件，注意**低版本的软件不支持我们这款芯片**。安装步骤简单，按提示安装即可。

2、硬件连接。

①使用串口前先用跳帽将 3.3v 与 VCC 连接。然后使用杜邦线将串口与开发板连接，具体为：串口 TXD 与开发板 MCU-RXD1 相连，串口 RXD 与开发板 MCU-TXD1 相连，串口 GND 与开发板 GND 相连。液晶屏左侧插针与右侧文字标识一一对应。串口连接如图 1- 32 所示：

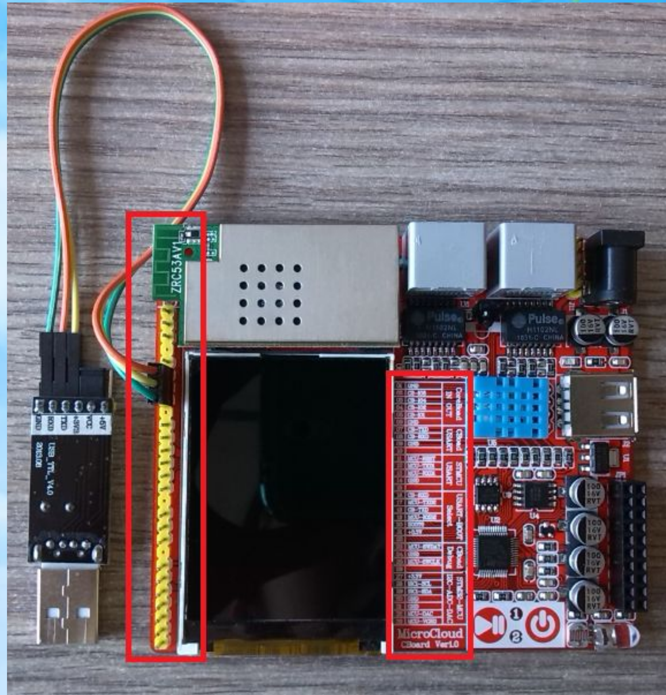


图 1- 32 串口连接

②**用跳帽按**图 1- 33 **所示连接**，使得 boot0=1 即 boot0 引脚接 3.3V；

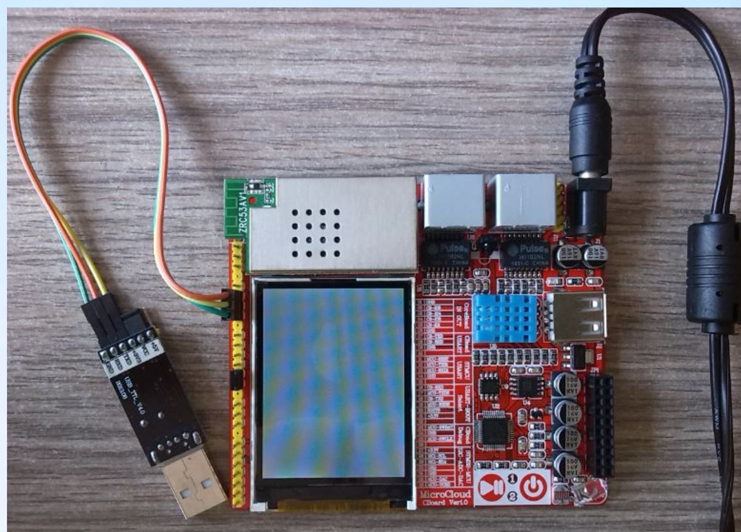


图 1- 33boot0 硬件连接

3、Flash_Loader_Demonstrator_v2.6.0 的使用。

前面两步骤完成后，插上串口及电源，将开发板上电。然后打开 Flash_Loader_Demonstrator_v2.6.0。进入使用界面。

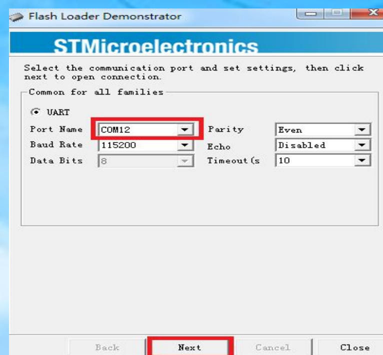


图 1- 34 选择串口号

选好串口号，点击 NEXT。如果**先打开的软件后给开发板上电，则需要将开发板重新上电**（开发板 RST 与地相接一下，再拔下。RST 的具体位置如图 1- 35 所示），

然后再点击 NEXT。否则将出现图 1- 36 的现象。

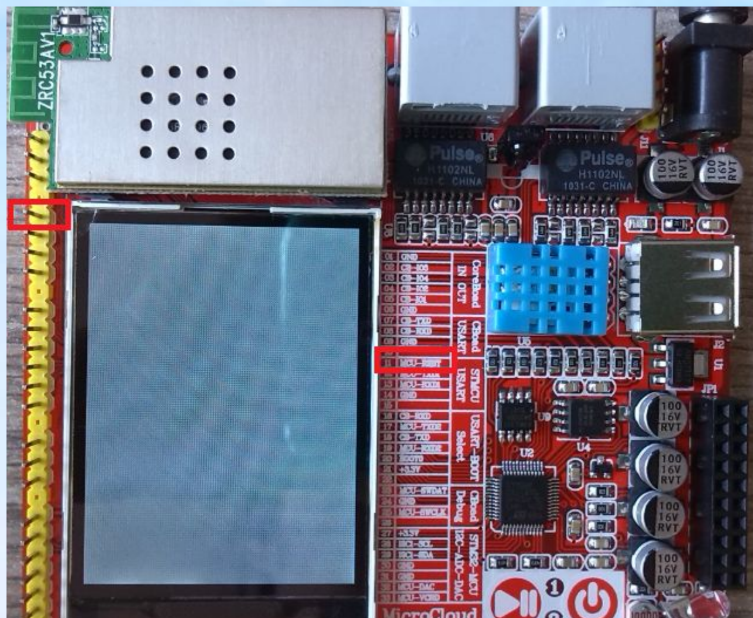


图 1- 35RST 引脚位置

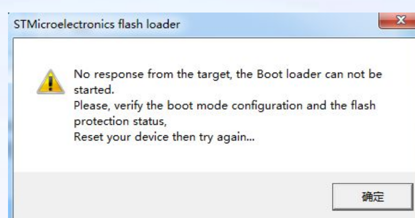


图 1- 36 错误信息

如果无误，将进入图 1- 37

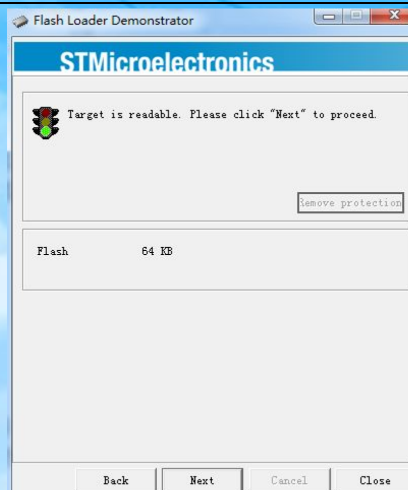


图 1- 37 目标芯片自动获取

选择默认，一路 next。出现如下界面图 1- 38：

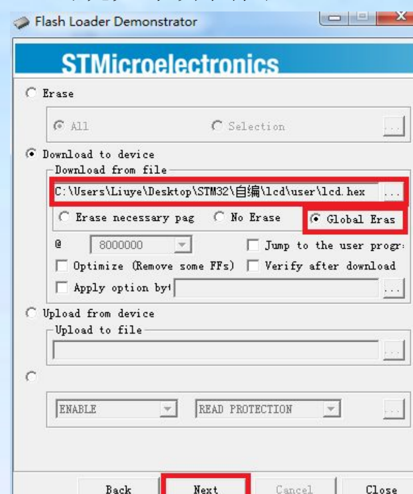
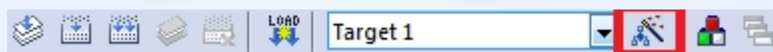


图 1- 38 选择 hex 文件

选择 Download to device，Global Eras，选择要下载的 hex 文件，目录在该工程的 user 文件夹下。如果没有要下载的 hex 文件，打开所要下载的工程，在工具栏处 点击如下按钮：



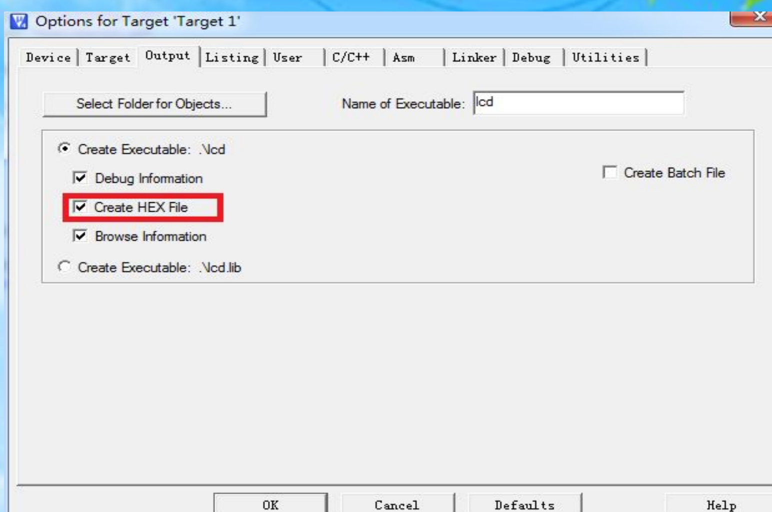


图 1- 39 生成 hex 文件

按图 1- 39 所示勾选，最后确认即可。

再选择该 hex 文件，最后点击 next 开始下载。

出现下面的图 1- 40 即表示下载完成：

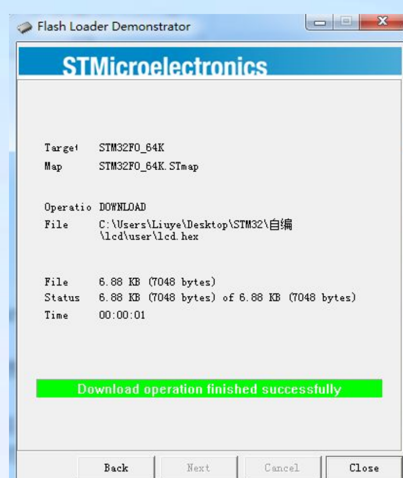


图 1- 40 串口下载成功

点 Close 关闭程序，关闭开发板电源，BOOT0 选择为 0，即将跳帽拔下，在重新插上开发板电源，即**在开发板上看到现象**。

注意：若要再次下载程序，请先关掉开发板电源，关闭 Flash_Loader_Demonstrator，然后重复上面步骤。

4.4.2 ST-LINK 下载

ST-LINK V2 支持 JTAG 和 SWD，同时 STM32 也支持 JTAG 和 SWD。所以，我们有 2 种方式可以用来调试，JTAG 调试的时候，占用的 IO 线比较多，而 SWD 调试的时候占用的 IO 线很少。

在使用 ST-LINK 之前请先**安装 ST-LINK V2 驱动程序**。安装步骤简单。

我们接上 ST-LINK V2，并把 SWD，SWCLK，GND 口插到 STM32+linux 开发板上，硬件连接如图 1- 41 下所示，注意一定要先连接仿真器，再上电，否则容易

烧坏仿真器。

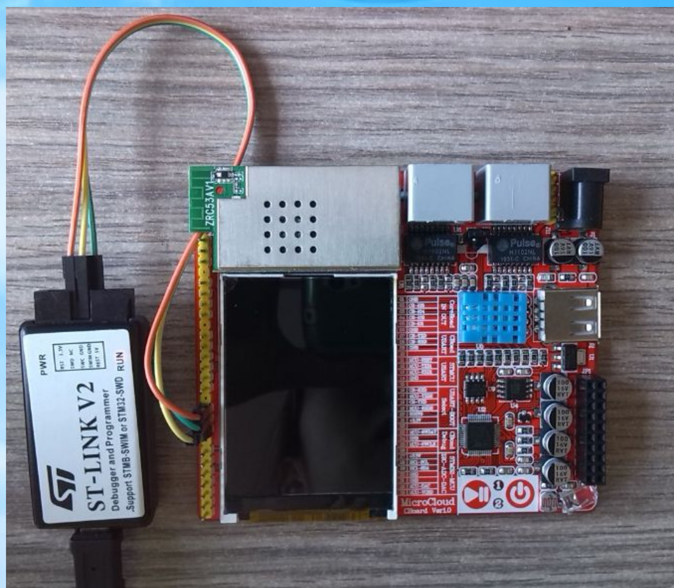


图 1- 41 仿真器连接

仿真器 SWD 连接开发板 MCU-SWDAT, SWC 连接开发板 MCU-SWCLK, GND 连接开发板 GND。

打开之前新建的工程，点击，打开 Options for Target 选项卡，如图 1- 42 所示：

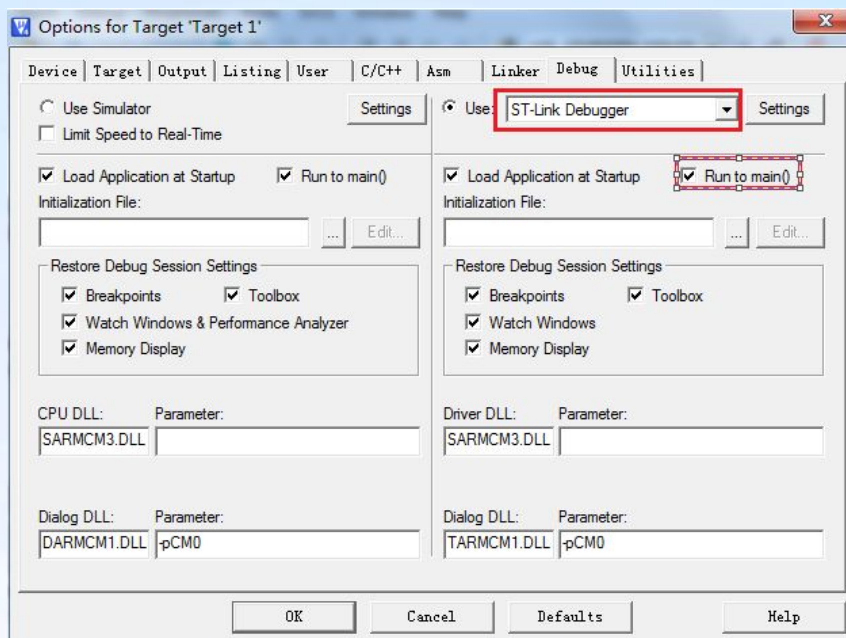


图 1- 42 选择 ST-LINK Debugger

左边的 Use Simulator 是进行软件仿真，即不连接开发板，只是单纯的用软件调试程序，右边是硬件仿真和下载程序的设置，我们将选择硬件仿真，在 Debug 栏选择仿真工具为 ST-LINK Debugger，同时勾选了 Run to main()，该选项选中后，只要点击仿真就会直接运行到 main 函数，如果没选择这个选项，则会先执行 startup_stm32f0xx.s 文件的 Reset_Handler，再跳到 main 函数。

然后我们点击 Settings，设置 ST-LINK 的一些参数，如图 1- 43 所示：



图 1- 43 设置 ST-LINK 参数

单击确定，完成此部分设置，接下来我们还需要在 Utilities 选项卡里面设置下载时的目标编程器，如图 1- 44 所示：



图 1- 44 设置目标编程器

我们选择 ST-LINK 来调试 Cortex M3，然后点击 Settings，设置如图 1- 45 所示：

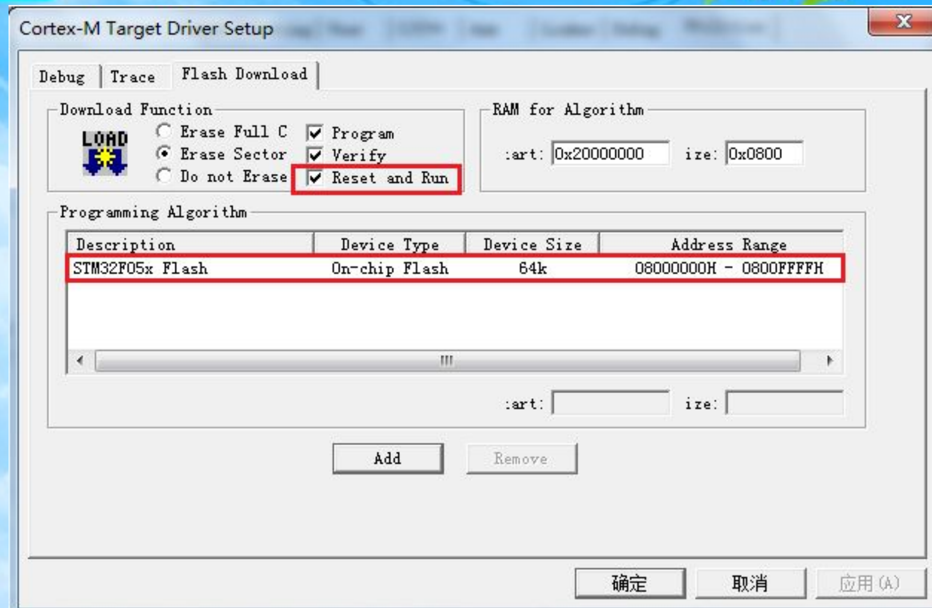


图 1-45 选择 FLASH

这里要根据不同的 MCU 选择 FLASH 的大小，因为我们开发板使用的是 STM32F051R8，其 FLASH 大小为 64KB，所以我们点击 Add，并 Programming Algorithm 里面选择 64K 型号的 STM32。然后选中 Reset and Run 选项，以实现在编程后自动启动，其他默认设置即可。在设置完之后，点击确定，然后再点击 OK，回到 IDE 界面，点击 编译一下（如果是自己新建的工程，编译有错误的话，可能是配置有误，如上所讲的配置一下）。再点击 ，开始硬件仿真，你也可以通过按 ，只下载代码，而不进入仿真。

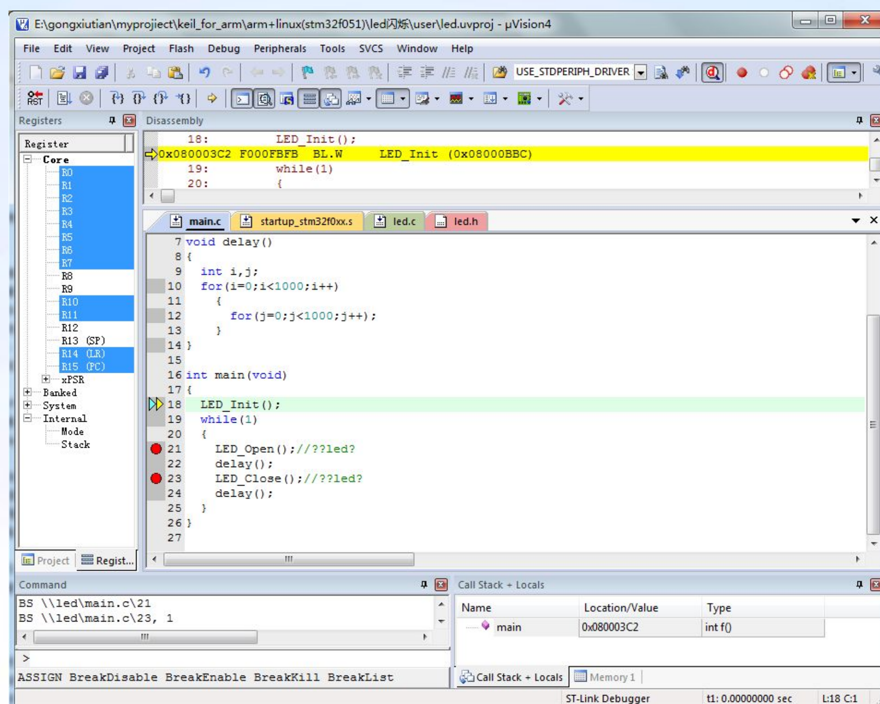


图 1-46 下载代码

将程序下载进去后，可以看到编译的窗口信息如下：

```
FromELF: creating hex file...
"led.axf" - 0 Error(s), 0 Warning(s).
Load "E:\\gongxiutian\\myprojiect\\keil_for_arm\\arm+linux(stm32f051)\\led闪烁\\user\\led.axf"
Erase Done.
Programming Done.
Verify OK.
Application running ...
```

同时也可以发现，多出了一个工具条，这就是 Debug 工具条，这个工具条在我们仿真的时候是非常有用的，下面简单介绍一下 Debug 工具条相关按钮的功能。Debug 工具条部分按钮的功能如图 1- 30 所示：

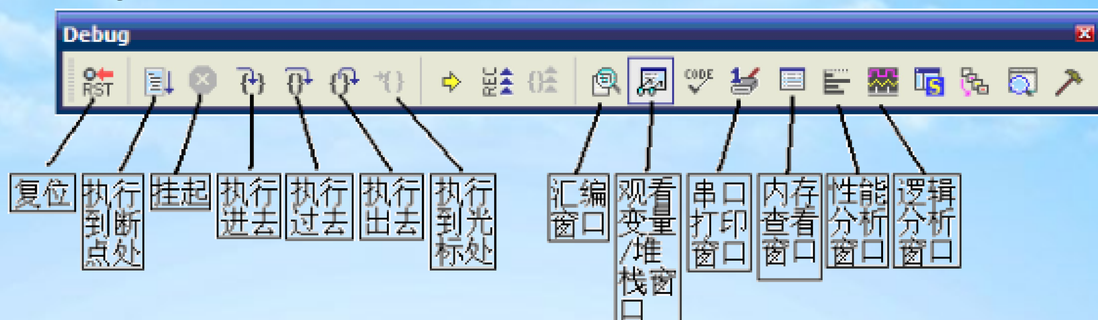


图 1- 47 工具条按钮

复位：其功能等同于硬件上按复位按钮。相当于实现了一次硬复位。按下该按钮之后，代码会重新从头开始执行。

执行到断点处：该按钮用来快速执行到断点处，有时候你并不需要观看每步是怎么执行的，而是想快速的执行到程序的某个地方看结果，这个按钮就可以实现这样的功能，前提是你在查看的地方设置了断点。

挂起：此按钮在程序一直执行的时候会变为有效，通过按该按钮，就可以使程序停止下来，进入到单步调试状态。

执行进去：该按钮用来实现执行到某个函数里面去的功能，在没有函数的情况下，是等同于执行过去按钮的。

执行过去：在碰到有函数的地方，通过该按钮就可以单步执行过这个函数，而不进入这个函数单步执行。

执行出去：该按钮是在进入了函数单步调试的时候，有时候你可能不必再执行该函数的剩余部分了，通过该按钮就直接一步执行完函数余下的部分，并跳出函数，回到函数被调用的位置。

执行到光标处：该按钮可以迅速的使程序运行到光标处，其实是挺像执行到断点处按钮功能，但是两者是有区别的，断点可以有多个，但是光标所在处只有一个。

汇编窗口：通过该按钮，就可以查看汇编代码，这对分析程序很有用。

观看变量/堆栈窗口：该按钮按下，会弹出一个显示变量的窗口，在里面可以查看各种你想要看的变量值，也是很常用的一个调试窗口。

串口打印窗口：该按钮按下，会弹出一个类似串口调试助手界面的窗口，用来显示从串口打印出来的内容。

内存查看窗口：该按钮按下，会弹出一个内存查看窗口，可以在里面输入你要查看的内存地址，然后观察这一片内存的变化情况。是很常用的一个调试窗口

性能分析窗口：按下该按钮，会弹出一个观看各个函数执行时间和所占百分比的窗口，用来分析函数的性能是比较有用的。

逻辑分析窗口：按下该按钮会弹出一个逻辑分析窗口，通过 SETUP 按钮新建一些 IO 口，就可以观察这些 IO 口的电平变化情况，以多种形式显示出来，比较直观。

Debug 工具条上的其他几个按钮用的比较少，我们这里就不介绍了。以上介绍的是比较常用的，当然也不是每次都用得着这么多，具体看你程序调试的时候有没有必要观看这些东西，来决定要不要看。

至此，我们就将 STM32F0 的入门知识介绍完毕了，接下来我们将按照实际的例子，来一点点的学习它。

第 5 章 STM32 初探—点亮 Led

5.1STM32 IO 简介

本章要实现的是控制微云 STM32+linux 开发板上的 led 灯闪烁的效果。该实验的关键在于如何控制 STM32 的 IO 的输出。因为本章是第一个实验章节，所以我们在这一章将讲解一些知识为后面的实验做铺垫。

在介绍 STM32 的 GPIO 之前，首先打开我们资料包的第一个固件库版本实验工程跑马灯实验工程(目录为:..\stm32+linux 模块\STM32+linux 资料包\例程\led 闪烁\user\led.Uv2”）,可以看到我们的实验工程目录如图 2- 1：

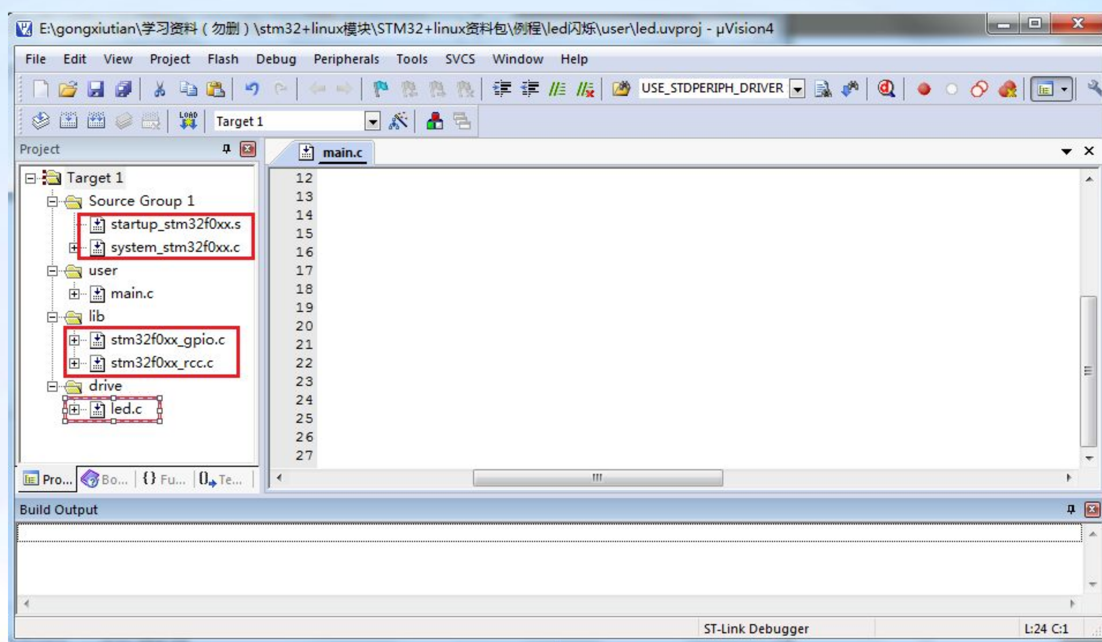


图 2- 1 跑马灯实验工程

接下来我们逐一讲解一下我们的工程目录下面的组以及重要文件。

①组 drive 下面存放的是外设驱动代码，它通过调用 FWLib 下面的固件库文件实现的，比如 led.c 里面调用 stm32f0xx_gpio.c 里面的函数对 led 进行初始化，这里面的函数是讲解的重点。后面的实验中可以看到会引入多个源文件。

②组 USER 下面存放的主要是用户代码。Main.c 函数主要存放的是主函数了，这个大家应该很清楚。

③组 lib 下面存放的是 ST 官方提供的固件库函数，里面的函数我们可以根据需要添加和删除，但是一定要注意在头文件 stm32f0xx_conf.h 文件中注释掉删除的源文件对应的头文件，这里面的文件内容用户不需要修改。

④组 Source Group 1 下面存放的是固件库必须的核心文件和启动文件。system_stm32f0xx.c 文件用户不需要修改。

针对第③步中怎么随意添加和删除固件库文件，这里我们稍微讲解一下。

首先从下面的图 2- 2 中可以看到，stm32f0xx_gpio.c 源文件下面 include 了好几个头文件，其中有一个 stm32f0xx_conf.h，这个文件会被每个固件库源文件引用。我们可以双击打开看看里面的内容：

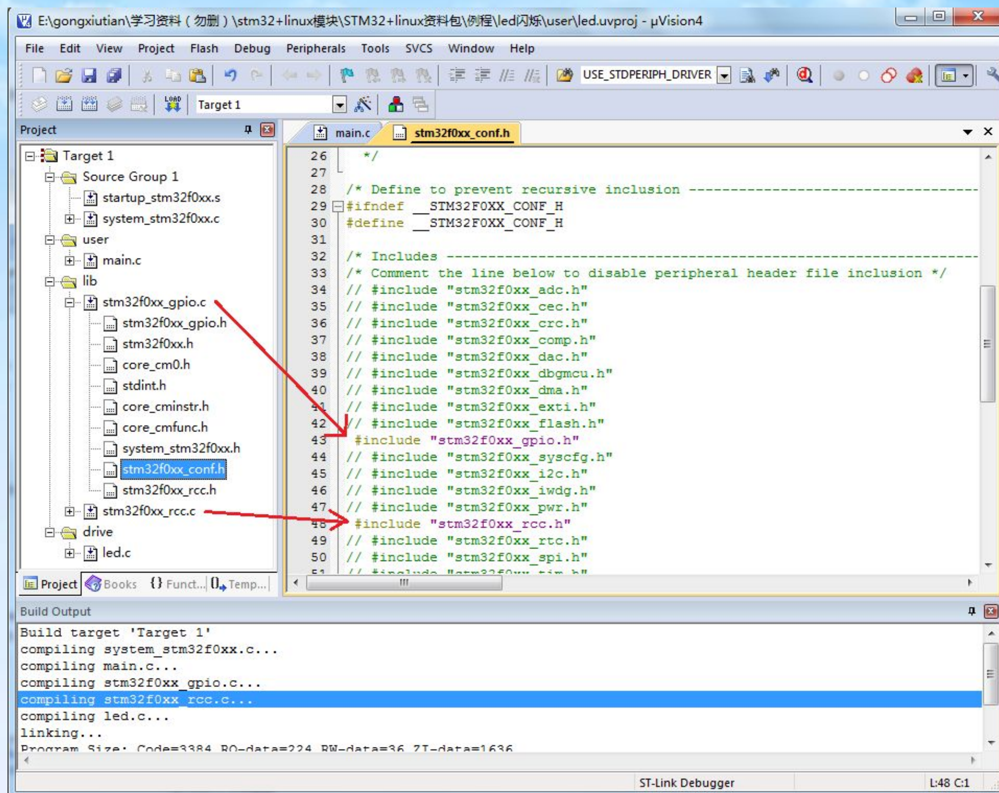


图 2- 2 stm32f0xx_conf.h

从图 2- 2 中可以看出，在头文件 stm32f0xx_conf.h 文件中，我们包含了两个.h 头文件，那是因为我们的 lib 组下面引入了相应的 2 个.c 源文件。同时大家记住，后面一个源文件 stm32f0xx_rcc.c,在每个实验基本都需要添加。在这个实验中，因为 LED 是关系到 STM32 的 GPIO，所以我们增加了 stm32f0xx_gpio.c 和头文件 stm32f0xx_gpio.h 的引入。添加和删除固件库源文件的步骤是：

1. 在 stm32f0xx_conf.h 文件引入需要的.h 头文件。这些头文件在每个实验的目录\lib\inc 下面都有存放。

2. 在 lib 下面加入步骤一中引入的.h 头文件对应的源文件。记住最好一一对应，否则就有可能会报错。这些源文件在每个实验的\lib\src 目录下面都有存放。添加方法请参考上一章的内容。

准备内容我们就讲解到这里，接下来我们就要进入我们 led 灯闪烁实验的讲解部分了。在固件库中，GPIO 端口操作对应的库函数函数以及相关定义在文件



stm32f0xx_gpio.h 和 stm32f0xx_gpio.c 中。STM32 的 IO 口相比 51 而言要复杂得多，所以使用起来也困难很多。

在配置 IO 口之前我们需要先使能 IO 口的时钟，其实现的函数为：

```
void RCC_AHBPeriphClockCmd(uint32_t RCC_AHBPeriph, FunctionalState NewState)
```

IO 口可以配置为 4 种模式：输入模式，输出模式，复用模式，模拟通道模式。由于 stm32f051 系列多数的 IO 关键复用了其外设功能，比如 I2C,SPI,UART 等，此时就可以设置 IO 口为复用模式。模拟通道则作为 AD,DA 的时候使用：

位 2y+1:2y　MODERy[1:0]: 端口 x 配置位 (y = 0..15)
　　这些位可由软件写来配置 I/O 口模式。
　　00: 输入模式 (复位状态)
　　01: 通用输出模式
　　10: 复用功能模式
　　11: 模拟模式

如果大家使用库函数编程的时候，可以在 stm32f0xx_gpio.h 文件中找到设置 IO 模式的结构体 GPIOMode_TypeDef ：

```
typedef enum
{
  GPIO_Mode_IN   = 0x00, /*!< GPIO Input Mode              */
  GPIO_Mode_OUT  = 0x01, /*!< GPIO Output Mode             */
  GPIO_Mode_AF   = 0x02, /*!< GPIO Alternate function Mode */
  GPIO_Mode_AN   = 0x03  /*!< GPIO Analog In/Out Mode      */
}GPIOMode_TypeDef;
```

这里我们点亮一个小灯需要将 IO 口设置成输出模式，而输出模式有两种，开漏输出和推挽输出。在 stm32f0xx_gpio.h 文件中找到设置输出模式的结构体 GPIOOType_TypeDef ：

位 31:16　保留，必须保持为复位值。
位 15:0　OTy[1:0]: 端口 x 的配置位 (y = 0..15)
　　这些位可由软件写来配置 I/O 口的输出类型。
　　0: 推挽输出 (复位状态)
　　1: 开漏输出

```
typedef enum
{
  GPIO_OType_PP = 0x00,
  GPIO_OType_OD = 0x01
}GPIOOType_TypeDef;
```

同时我们可以设置 IO 端口输出的速度，在库函数中通过结构体来解决：

位 2y+1:2y　OSPEEDRy[1:0]: 端口 x 配置位 (y = 0..15)
　　这些位要由软件写来配置 I/O 口的速度。
　　x0: 低速
　　01: 中速
　　11: 高速

```
typedef enum
{
  GPIO_Speed_Level_1  = 0x01, /*!< Medium Speed */
  GPIO_Speed_Level_2  = 0x02, /*!< Fast Speed   */
  GPIO_Speed_Level_3  = 0x03  /*!< High Speed   */
}GPIOSpeed_TypeDef;
```

对于 IO 端口的配置，我们通过函数 GPIO_Init()来完成上述对 IO 口的配置，



此函数的原型如下所示：

```
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
```

其中 GPIOx 是需要配置的端口，GPIO_InitStruct 是需要配置的端口的结构体：原型如下：

```
typedef struct
{
  uint32_t GPIO_Pin;              /*!< Specifies the GPIO pins to be configured.
                                       This parameter can be any value of @ref GPIO_pins_define */

  GPIOMode_TypeDef GPIO_Mode;     /*!< Specifies the operating mode for the selected pins.
                                       This parameter can be a value of @ref GPIOMode_TypeDef   */

  GPIOSpeed_TypeDef GPIO_Speed;   /*!< Specifies the speed for the selected pins.
                                       This parameter can be a value of @ref GPIOSpeed_TypeDef  */

  GPIOOType_TypeDef GPIO_OType;   /*!< Specifies the operating output type for the selected pins.
                                       This parameter can be a value of @ref GPIOOType_TypeDef  */

  GPIOPuPd_TypeDef GPIO_PuPd;     /*!< Specifies the operating Pull-up/Pull down for the selected pins.
                                       This parameter can be a value of @ref GPIOPuPd_TypeDef   */
}GPIO_InitTypeDef;
```

从结构体里可以看出，对 IO 口的初始化配置里包括了上述对 IO 口的模式，输出类型以及输出速率的配置。

最后我们对 IO 口的引脚拉高或者拉低，便可以实现 led 灯的闪烁了，其实现的函数分别为：

```
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

GPIO 相关的函数我们先讲解到这里。虽然 IO 操作步骤很简单，这里我们还是做个概括性的总结，操作步骤为：

1） 使能 IO 口时钟。调用函数为 RCC_APB2PeriphClockCmd()。
2） 初始化 IO 参数。调用函数 GPIO_Init();
3） 操作 IO。操作 IO 的方法就是上面我们讲解的方法。

上面我们讲解了 STM32 IO 口的基本知识以及固件库操作 GPIO 的一些函数方法，下面我们来讲解我们的 led 实验的硬件和软件设计。

5.2硬件电路原理

本节用到的硬件只有 LED（D1）。其电路在开发板上默认是已经连接好了的。所以无需外接杜邦线。硬件原理图如图 2- 3 下所示：



图 2- 3LED 硬件连接

由图 2- 3 可以看出，LED 引脚置高灯亮，LED 引脚置低灯灭。由 STM32 控制的引脚为 PA11 口，如下图 2- 4 所示：

图 2- 4STM32 控制引脚

5.3软件程序与注解

双击打开我们资料包里的 LED 闪烁的例程（路径为..\STM32+linux 资料包\例程\led 闪烁\user\led.Uv2），如图 2- 5 可以看到如下的工程：

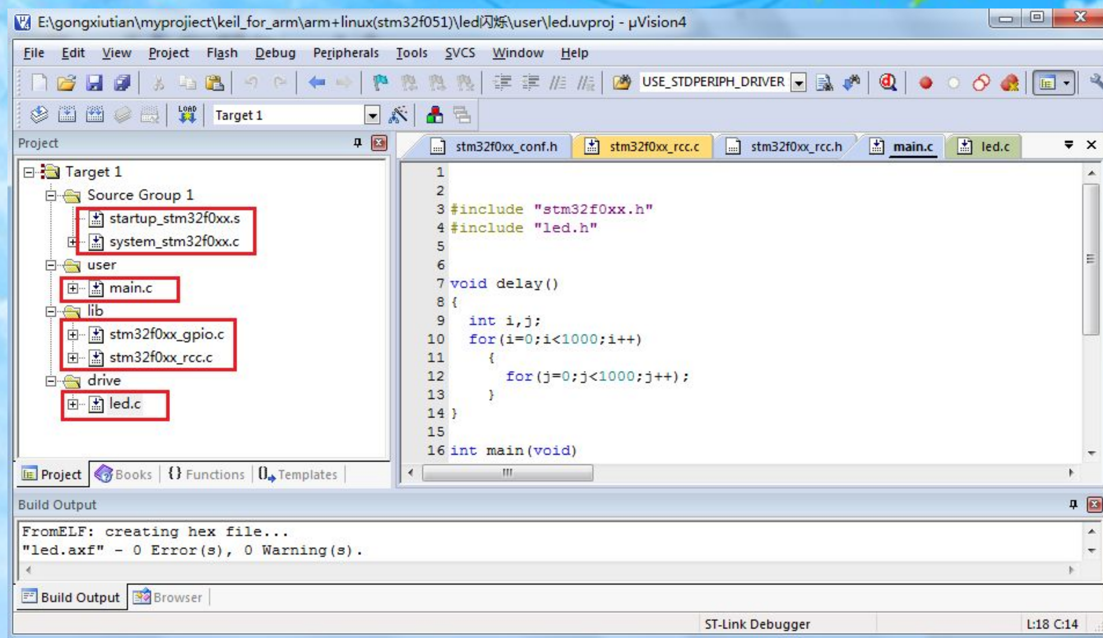


图 2- 5LED 闪烁的例程

工程包含四个组，Source Group1 是新建工程时默认的一个组，我们就把启动文件 startup_stm32f0xx.s 和核心文件 system_stm32f0xx.c 放到这个组里，这两个文件基本上每个工程都要添加，不需要作任何改动；lib 里存放的是固件库里的文件，我们这里只在工程里添加了 stm32f0xxx_gpio.c 和 stm32f0xxx_rcc.c，两个库文件，其中 stm32f0xxx_gpio.c 是驱动 IO 口的文件，stm32f0xxx_.rcc.c 是和系统时钟相关的文件，基本上也是每个工程都需要添加的，因为不论是什么外设，都需要启动时钟的。剩下两个组里的 main.c 和 led.c 是我们用户编辑的驱动文件，先看主函数，它包括的头文件有：

```
#include "stm32f0xx.h"
#include "led.h"
```

stm32f0xx.h 是系统文件，每个工程都必须包含的文件，就像 51 单片机里的 reg51.h 一样；led.h 是我们编辑的另外一个文件 led.c 的头文件，此头文件里包括的对 led 对应的引脚的定义以及 led.c 文件里函数的声明。如图 2- 6 我们将光标移到 led.h 的位置，右击选择 Open Document ‘led.h’，打开 led.h 文件：

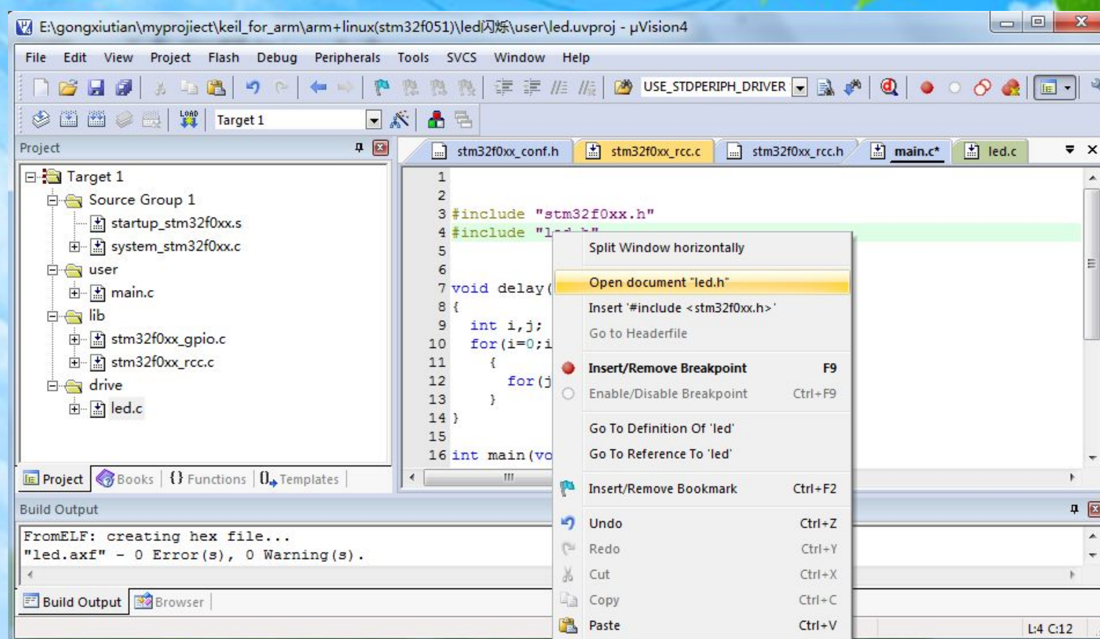


图 2- 6 打开 led.h 文件

可以看到如下所示：

```
#ifndef __LED_H
#define __LED_H

#include "stm32f0xx.h"
#define LED_PIN       GPIO_Pin_12
#define LED_PORT        GPIOA
void LED_Init(void);
void LED_Open(void);
void LED_Close(void);
void LED_Toggle(void);
#endif /* __LED_H */
```

此文件里我们定义了 led 灯的引脚为 GPIO 的第 12 个引脚，定义了 led 灯的端口为 GPIOA 端口，同时声明了四个和 led 有关的函数（函数的详细说明请看下文）。

我们回到主函数里，接下来是一个与 led 灯的亮灭相结合的延时函数：

```
void delay()
{
  int i,j;
  for(i=0;i<1000;i++)
    {
      for(j=0;j<1000;j++);
    }
}
```

然后是主函数：



```
int main(void)
{
  LED_Init();
  while(1)
  {
    LED_Open();//点亮led灯
    delay();
    LED_Close();//关闭led灯
    delay();
  }
}
```

首先对 led 灯进行了初始化，如图 2- 7 我们将光标移到这个函数，然后右击鼠标，选择 Go To Definition Of‘LED_init’调到此函数的定义部分，此函数定义在 led.c 文件里：

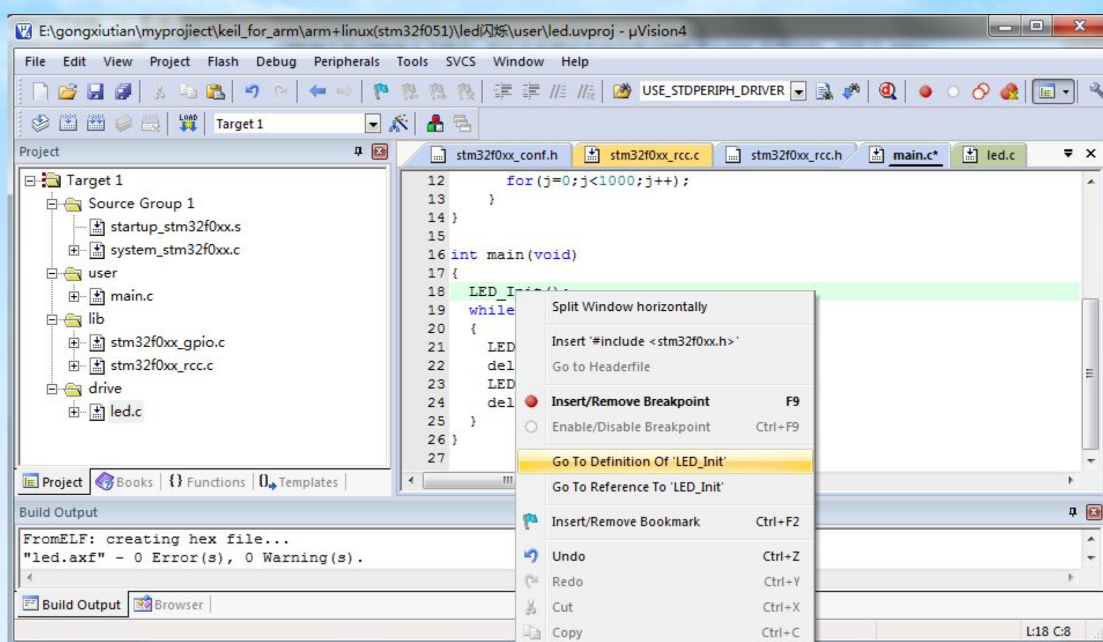


图 2- 7 Go To Definition Of‘LED_init’

函数的定义如下所示：

```
void LED_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
  GPIO_InitStruct.GPIO_Pin = LED_PIN  ;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Speed =GPIO_Speed_Level_3;
  GPIO_Init(LED_PORT, &GPIO_InitStruct);
  GPIO_SetBits(LED_PORT, LED_PIN  );
}
```

首先定义了一个 GPIO_InitTypeDef 结构体（GPIO_InitTypeDef 的解释请参看第一节内容），名字为 GPIO_InitStruct，用来存放 led 初始化的配置，然后用 RCC_AHBPeriphClockCmd()函数使能 GPIO 的时钟；将 LED 对应的引脚配置成推挽输出模式，最后将该引脚拉高 GPIO_SetBits(LED_PORT, LED_PIN);

我们再次回到主函数里，看 while(1)里的循环：




```
while(1)
{
  LED_Open();//点亮led灯
  delay();
  LED_Close();//关闭led灯
  delay();
}
```

从字面上可以看出是打开 led 灯，延时一段时间然后关闭 led 灯，再延时一会儿，我们用 Go To Definition Of ‘LED_init’跳到函数的相应位置，可以知道，打开 led 灯和关闭 led 灯实际上就是将 led 引脚拉高或者拉低：

```
/************************************************/
/* 函数功能：打开led灯                          */
/* 入口参数：无                                 */
/************************************************/
void LED_Open(void)
{
    GPIO_SetBits(LED_PORT, LED_PIN );
}
/************************************************/
/* 函数功能：  关掉led灯                         */
/* 入口参数：无                                 */
/************************************************/
void LED_Close(void)
{
    GPIO_ResetBits(LED_PORT, LED_PIN );
}
```

led.c 里还包含有另外一个函数就是 led 翻转函数：

```
/************************************************/
/* 函数功能：led翻转                            */
/* 入口参数：无                                 */
/************************************************/
void LED_Toggle(void)
{
GPIO_WriteBit(GPIOA, GPIO_Pin_12,
                   (BitAction)((1-GPIO_ReadOutputDataBit(GPIOA, GPIO_Pin_12))));
}
```

此函数就是将当前的引脚的状态读进来，然后用 1 减掉（1-0=1,1-1=0，实现翻转），再重新写回去，从而达到翻转的目的，此函数涉及到了其他的固件库函数，有兴趣的童鞋可以自己去翻看一下，这里就不再细说了。

至此我们就将 led 灯闪烁的工程讲解完毕，接下来我们将工程编译一下，下载到开发板里看一下现象。

5.4程序下载与测试

本工程实现的功能为：led 灯闪烁。

首先将微云 STM32+linux 开发板的串口连接好（跳帽将 3.3v 与 VCC 连接。然后使用杜邦线将串口与开发板连接，具体为：串口 TXD 与开发板 MCU-RXD1 相连，串口 RXD 与开发板 MCU-TXD1 相连，串口 GND 与开发板 GND 相连），然后给开发板上电

将程序下载进去，具体步骤参照串口下载，我们就可以看到板子上的小灯在

闪烁了，如图 2- 8 所示：

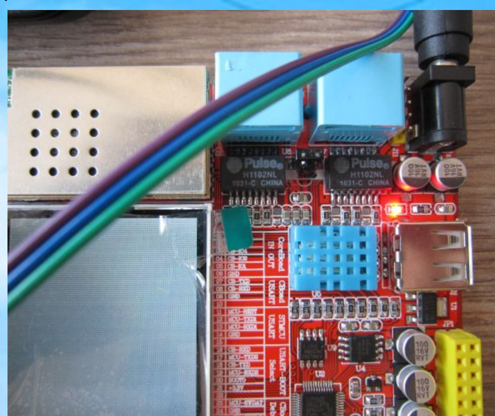


图 2- 8 小灯在闪烁

也可以通过仿真器仿真，仿真的步骤请参看 ST-LINK 下载。

至此，我们就将 led 闪烁的工程讲解完了，虽然这个工程比较简单，但是因为是入门，同时也是基础，所以讲的比较详细一些，希望学习的童鞋好好消化，为以后的学习做好准备。

第 6 章 串口收发功能实验

6.1STM32 串口 USART 简介

串口作为 MCU 的重要外部接口，同时也是软件开发重要的调试手段，其重要性不言而喻。现在基本上所有的 MCU 都会带有串口，STM32 自然也不例外。STM32 的串口资源相当丰富的，功能也相当强劲。STM32F051 设有两个串口 USART1 和 USART2，有分数波特率发生器、支持同步单线通信和半双工单线通讯、支持 LIN、支持调制解调器操作、智能卡协议和 IrDA SIR ENDEC 规范、具有 DMA 等。

接下来我们将主要从库函数操作层面结合寄存器的描述，告诉你如何设置串口，以达到我们最基本的通信功能。本章，我们将实现利用 UASRT1 不停的打印信息到电脑上，同时接收从串口发过来的数据，把发送过来的数据直接送回给电脑。战舰 STM32 开发板板载了 1 个 USB 串口和 1 个 RS232 串口，我们本章介绍的是通过 USB 串口和电脑通信。

使用串口首先要将对应的 IO 口设置成端口复用模式，对于复用功能的 IO，我们首先要使能 GPIO 时钟，然后使能复用功能时钟，同时要把 GPIO 模式设置为复用功能对应的模式（这个可以查看手册《02_STM32F05x8 参考手册（中文）》“外设的 GPIO 配置”）。这些准备工作做完之后，剩下的当然是串口参数的初始化设置，包括波特率，停止位等等参数。在设置完成只能接下来就是使能串口，这很容易理解。同时，如果我们开启了串口的中断，当然要初始化 NVIC 设置中断优先级别，最后编写中断服务函数。

串口设置的一般步骤可以总结为如下几个步骤：

1) 串口时钟使能，GPIO 时钟使能
2) 串口复位
3) GPIO 端口模式设置

4) 串口参数初始化
5) 开启中断并且初始化 NVIC（如果需要开启中断才需要这个步骤）
6) 使能串口
7) 编写中断处理函数

下面，我们就简单介绍下这几个与串口基本配置直接相关的几个固件库函数。这些函数和定义主要分布在 stm32f0xx_usart.h 和 stm32f0xx_usart.c 文件中。

1. 串口时钟和 IO 口时钟使能。其中串口是挂载在 APB2 下面的外设，所以使能函数为：

```
RCC_AHBPeriphClockCmd( RCC_AHBPeriph_GPIOA, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE );
```

2. 串口复位。当外设出现异常的时候可以通过复位设置，实现该外设的复位，然后重新配置这个外设达到让其重新工作的目的。一般在系统刚开始配置外设的时候，都会先执行复位该外设的操作。复位的是在函数 USART_DeInit()中完成：

```
void USART_DeInit(USART_TypeDef* USARTx)
```

比如我们要复位串口 1，方法为：

```
USART_DeInit(USART1);
```

3. 串口参数初始化。串口初始化是通过 USART_Init()函数实现的，

```
void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)
```

这个函数的的第一个入口参数是指定初始化的串口标号，这里选择 USART1。第二个入口参数是一个 USART_InitTypeDef 类型的结构体指针，这个结构体指针的成员变量用来设置串口的一些参数。一般的实现格式为：

```
USART_InitStructure.USART_BaudRate = 115200;//设置串口波特率
USART_InitStructure.USART_WordLength = USART_WordLength_8b;//设置数据位
USART_InitStructure.USART_StopBits = USART_StopBits_1;//设置停止位
USART_InitStructure.USART_Parity = USART_Parity_No;//设置效验位
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;//设置流控制
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;//设置工作模式
USART_Init(USART1, &USART_InitStructure); //配置入结构体
```

从上面的初始化格式可以看出初始化需要设置的参数为：波特率，字长，停止位，奇偶校验位，硬件数据流控制，模式（收，发）。我们可以根据需要设置这些参数。

4. 数据发送与接收。STM32 的发送与接收是通过数据寄存器 USART_DR 来实现的，这是一个双寄存器，包含了 TDR 和 RDR。当向该寄存器写数据的时候，串口就会自动发送，当收到收据的时候，也是存在该寄存器内。

STM32 库函数操作 USART_DR 寄存器发送数据的函数是：

```
void USART_SendData(USART_TypeDef* USARTx, uint16_t Data)
```

通过该函数向串口寄存器 USART_DR 写入一个数据。

STM32 库函数操作 USART_DR 寄存器读取串口接收到的数据的函数是：

```
uint16_t USART_ReceiveData(USART_TypeDef* USARTx)
```

通过该函数可以读取串口接受到的数据。

5.串口状态。串口的状态可以通过中断和状态寄存器 USART_ISR 读取。USART_SR 的各位描述如图所示：



24.7.8 中断和状态寄存器(USART_ISR)

地址偏移：0x1C

复位值：0x00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res	RE ACK	TE ACK	WUF	RWU	SBKF	CMF	BUSY
									r	r	r	r	r	r	r
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ABRF	ABRE	ABRE	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

我们关注一下两个位，第 5、6 位 RXNE 和 TC。

RXNE（读数据寄存器非空），当该位被置 1 的时候，就是提示已经有数据被接收到了，并且可以读出来了。这时候我们要做的就是尽快去读取 USART_DR，通过读 USART_DR 可以将该位清零，也可以向该位写 0，直接清除。

TC（发送完成），当该位被置位的时候，表示 USART_DR 内的数据已经被发送完成了。如果设置了这个位的中断，则会产生中断。该位也有两种清零方式：1）读 USART_SR，写 USART_DR。2）直接向该位写 0。状态寄存器的其他位我们这里就不做过多讲解，大家需要可以查看中文参考手册。在我们固件库函数里面，读取串口状态的函数是：

```
FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint32_t USART_FLAG)
```

这个函数的第二个入口参数非常关键，它是标示我们要查看串口的哪种状态，比如上面讲解的 RXNE(读数据寄存器非空)以及 TC(发送完成)。例如我们要判断读寄存器是否非空(RXNE)，操作库函数的方法是：

```
USART_GetFlagStatus(USART1, USART_FLAG_RXNE);
```

我们要判断发送是否完成(TC)，操作库函数的方法是：

```
USART_GetFlagStatus(USART1, USART_FLAG_TC);
```

这些标志号在 MDK 里面是通过宏定义定义的:

```
#define USART_IT_WU                           ((uint32_t)0x00140316)
#define USART_IT_CM                           ((uint32_t)0x0011010E)
#define USART_IT_EOB                          ((uint32_t)0x000C011B)
#define USART_IT_RTO                          ((uint32_t)0x000B011A)
#define USART_IT_PE                           ((uint32_t)0x00000108)
#define USART_IT_TXE                          ((uint32_t)0x00070107)
#define USART_IT_TC                           ((uint32_t)0x00060106)
#define USART_IT_RXNE                         ((uint32_t)0x00050105)
#define USART_IT_IDLE                         ((uint32_t)0x00040104)
#define USART_IT_LBD                          ((uint32_t)0x00080206)
#define USART_IT_CTS                          ((uint32_t)0x0009030A)
#define USART_IT_ERR                          ((uint32_t)0x00000300)
#define USART_IT_ORE                          ((uint32_t)0x00030300)
#define USART_IT_NE                           ((uint32_t)0x00020300)
#define USART_IT_FE                           ((uint32_t)0x00010300)
```

6.串口使能。串口使能是通过函数 USART_Cmd()来实现的，这个很容易理解，使用方法是：

```
USART_Cmd(USART1, ENABLE);//使能串口1
```

7.开启串口响应中断。有些时候当我们还需要开启串口中断，那么我们还需要使

能串口中断，使能串口中断的函数是：

```
void USART_ITConfig(USART_TypeDef* USARTx, uint32_t USART_IT, FunctionalState NewState)
```

这个函数的第二个入口参数是标示使能串口的类型，也就是使能哪种中断，因为串口的中断类型有很多种。比如在接收到数据的时候（RXNE 读数据寄存器非空），我们要产生中断，那么我们开启中断的方法是：

```
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
```

8.获取相应中断状态。当我们使能了某个中断的时候，当该中断发生了，就会设置状态寄存器中的某个标志位。经常我们在中断处理函数中，要判断该中断是哪种中断，使用的函数是：

```
ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint32_t USART_IT)
```

比如我们使能了串口接收完成中断，那么当中断发生了， 我们便可以在中断处理函数中调用这个函数来判断到底是否是串口接收中断，方法是：

```
if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
```

If 语句为真，说明是串口接收中断发生。

通过以上的讲解，我们就可以达到串口最基本的配置了，关于串口更详细的介绍，请参考《02_STM32F05x8 参考手册（中文)》，通用同步异步收发器一章。

6.2STM32 NVIC 中断优先级管理

这节我们将对 STM32 的重要知识中断管理做个介绍。

STM32F051xx 有 48 个中断，包括 16 个内核中断和 32 个可屏蔽中断，具有 4 级可编程的中断优先级。

在 MDK 内，与 NVIC 相关的寄存器，MDK 为其定义了如下的结构体：

```
typedef struct
{
  __IO uint32_t ISER[1];                 /*!< Offset: 0x000 (R/W)
       uint32_t RESERVED0[31];
  __IO uint32_t ICER[1];                 /*!< Offset: 0x080 (R/W)
       uint32_t RSERVED1[31];
  __IO uint32_t ISPR[1];                 /*!< Offset: 0x100 (R/W)
       uint32_t RESERVED2[31];
  __IO uint32_t ICPR[1];                 /*!< Offset: 0x180 (R/W)
       uint32_t RESERVED3[31];
       uint32_t RESERVED4[64];
  __IO uint32_t IP[8];                   /*!< Offset: 0x300 (R/W)
}  NVIC_Type;
```

STM32 的中断在这些寄存器的控制下有序的执行的。只有了解这些中断寄存器，才能了解 STM32 的中断。下面简要介绍这几个寄存器：

ISER[1]：ISER 全称是：Interrupt Set-Enable Registers，这是一个中断使能寄存器组。上面说了 STM32F051 的可屏蔽中断只有 32 个，这里用了 1 个 32 位的寄存器，总共可以表示 32 个中断。ISER[0]的 bit0~bit31 分别对应中断 0~31。你要使能某个中断，必须设置相应的 ISER 位为 1，使该中断被使能(这里仅仅是使能，还要配合中断分组、屏蔽、IO 口映射等设置才算是一个完整的中断设置)。具体每一位对应哪个中断，请参考 stm32f0xx.h 里面的第 171 行处。

ICER[1]：全称是：Interrupt Clear-Enable Registers，是一个中断除能寄存器组。该寄存器组与 ISER 的作用恰好相反，是用来清除某个中断的使能的。其

对应位的功能，也和 ICER 一样。这里要专门设置一个 ICER 来清除中断位，而不是向 ISER 写 0 来清除，是因为 NVIC 的这些寄存器都是写 1 有效的，写 0 是无效的。具体为什么这么设计，请看《CM0 权威指南》第 447 页，NVIC 概览一章。

ISPR[1]：全称是：Interrupt Set-Pending Registers，是一个中断挂起控制寄存器组。每个位对应的中断和 ISER 是一样的。通过置 1，可以将正在进行的中断挂起，而执行同级或更高级别的中断。写 0 是无效的。

ICPR[1]：全称是：Interrupt Clear-Pending Registers，是一个中断解挂控制寄存器组。其作用与 ISPR 相反，对应位也和 ISER 是一样的。通过设置 1，可以将挂起的中断接挂。写 0 无效。

IPR[8：全称是：Interrupt Priority Registers，是一个中断优先级控制的寄存器组。

关于中断的优先级管理，详细情况请参考《02_STM32F05x8 参考手册（中文）》

接下来我们介绍如何使用库函数实现以上中断分组设置以及中断优先级管理，使得我们以后的中断设置简单化。NVIC 中断管理函数主要在 stm32f0xx_misc.c 文件里面。

最需要讲解的是中断初始化函数 NVIC_Init，其函数申明为：

void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct)

其中 NVIC_InitTypeDef 是一个结构体，我们可以看看结构体的成员变量：

```
typedef struct
{
  uint8_t NVIC_IRQChannel;              /*!< Specifies the IRQ channel
                                             This parameter can be a va
                                             (For the complete STM32 De
                                             please refer to stm32f0xx.

  uint8_t NVIC_IRQChannelPriority;      /*!< Specifies the priority lev
                                             in NVIC_IRQChannel. This p
                                             between 0 and 3.  */

  FunctionalState NVIC_IRQChannelCmd;   /*!< Specifies whether the IRQ
                                             will be enabled or disable
                                             This parameter can be set
} NVIC InitTypeDef;
```

NVIC_InitTypeDef 结构体中间有三个成员变量，这三个成员变量的作用是：

NVIC_IRQChannel：定义初始化的是哪个中断，这个我们可以在 stm32f0xx.h 中找到每个中断对应的名字。例如 USART1_IRQn。

NVIC_IRQChannelPriority：定义这个中断的优先级别。

NVIC_IRQChannelCmd：该中断是否使能。

比如我们要使能串口 1 的中断，同时设置抢占优先级为 0，初始化的方法是：

```
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPriority =0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

最后我们总结一下，中断优先级设置实际上就是对每个中断调用函数为 NVIC_Init();

6.3硬件电路原理

开发板串口 1 的引线已经外接出来，如图 3- 1 所示：

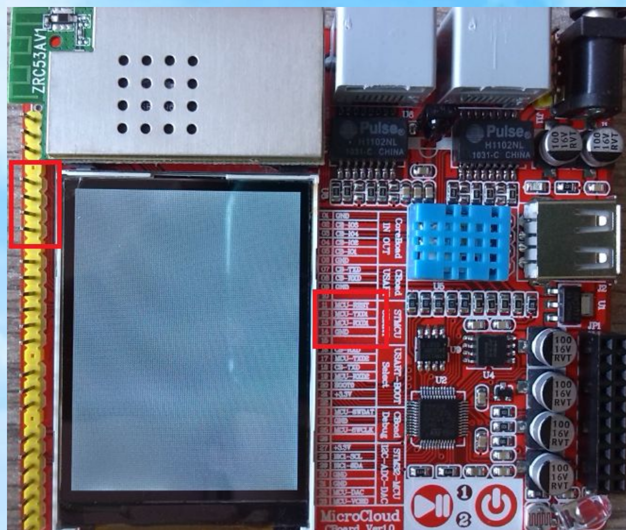


图 3- 1 串口硬件连接图

将串口与开发板串口 1 的连接（跳帽将串口的 3.3v 与 VCC 连接。然后使用杜邦线将串口与开发板连接，具体为：串口 TXD 与开发板 MCU-RXD1 相连，串口 RXD 与开发板 MCU-TXD1 相连，串口 GND 与开发板 GND 相连）

这里我们需要借助串口和电脑进行通信，详细操作步骤请参考本章第 5 节

6.4软件程序与注解

双击打开我们资料包里的串口收发测试的例程（路径为..\STM32+linux 资料包\例程\usart 串口（收发测试）\user\usart.Uv2），可以看到如图 3- 2 的工程：

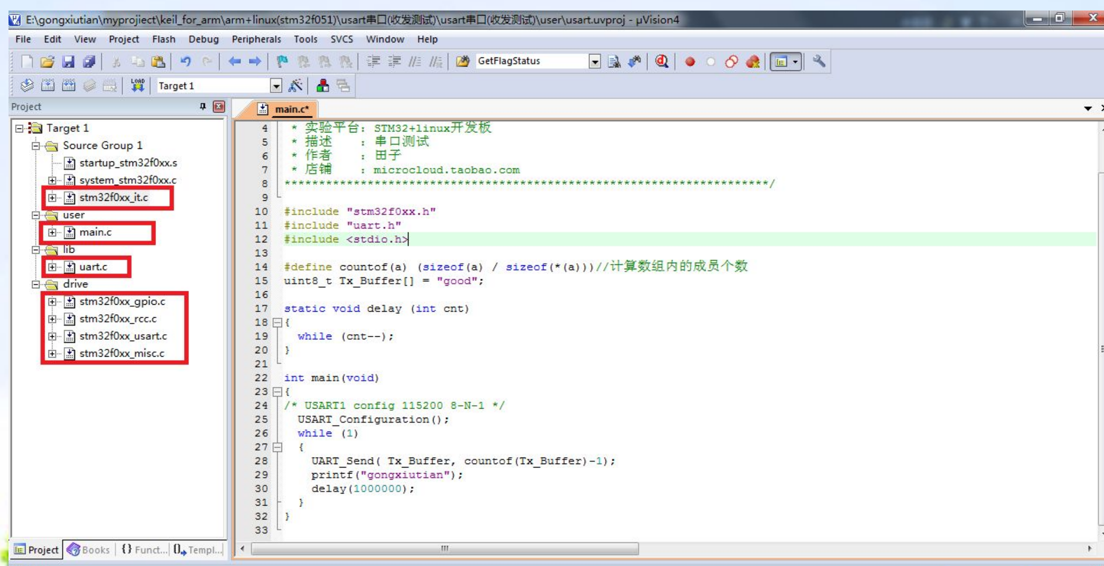


图 3- 2 串口工程

工程包含四个组，Source Group1 除了 startup_stm32f0xx.s 和 system_stm32f0xx.c 之外，还添加了一个编写中断函数的文件 stm32f0xx_it.c；lib



里存放的固件库里的文件里添加了四个库文件，其中 stm32f0xx_gpio.c 和 stm32f0xx_rcc.c 不细说。stm32f0xx_usart.c 是驱动串口的文件，stm32f0xx_misc.c 是配置中断需要的文件，剩下两个组里的 main.c 和 uart.c 是我们用户编辑的驱动文件，先看主函数，它包括的头文件有：

```
#include "stm32f0xx.h"
#include "uart.h"
#include <stdio.h>
```

然后是一个宏定义，和一个数组，以便向串口发送数据，以及一个简单的延时函数：

```
#define countof(a)  (sizeof(a) / sizeof(*(a)))//计算数组内的成员个数
uint8_t Tx_Buffer[] = "good";

static void delay (int cnt)
{
  while (cnt--);
}
```

然后是主函数：

```
int main(void)
{
/* USART1 config 115200 8-N-1 */
  USART_Configuration();
  while (1)
  {
    UART_Send( Tx_Buffer, countof(Tx_Buffer)-1);
    printf("welcome to use microcloud product!");
    delay(1000000);
  }
}
```

调到 USART_Configuration()函数(右击函数，Go To Definition of...)，可以看到如下代码：

```c
void USART_Configuration(void)//串口初始化函数
  {
        GPIO_InitTypeDef  GPIO_InitStructure;
        USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
  /*Enable the USART1 Interrupt(??USART1??)*/
      NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
      NVIC_InitStructure.NVIC_IRQChannelPriority =0;
      NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
      NVIC_Init(&NVIC_InitStructure);

        RCC_AHBPeriphClockCmd( RCC_AHBPeriph_GPIOA, ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE );
        GPIO_PinAFConfig(GPIOA,GPIO_PinSource9,GPIO_AF_1);
        GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_1);
        /*
        *  USART1_TX -> PA9 , USART1_RX ->        PA10
        */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9|GPIO_Pin_10;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
        GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOA, &GPIO_InitStructure);

        USART_InitStructure.USART_BaudRate = 115200;//设置串口波特率
        USART_InitStructure.USART_WordLength = USART_WordLength_8b;//设置数据位
        USART_InitStructure.USART_StopBits = USART_StopBits_1;//设置停止位
        USART_InitStructure.USART_Parity = USART_Parity_No;//设置效验位
        USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
        USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;//设置工作模式
        USART_Init(USART1, &USART_InitStructure); //配置入结构体
        USART_Cmd(USART1, ENABLE);//使能串口1
        USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    }
```

此段代码分为三部分，第一部分是设置串口的中断以及优先级，使能此中断；第二部分是配置 usart 的引脚为复用引脚；第三部分是对串口的设置，设置波特率为 115200,8 个数据位，无校验，开启发送和接收模式，最后使能串口以及串口接收中断。

我们回到主函数配置完串口之后，就进入循环，开始调用发送函数,发送字符“good”：

```c
UART_Send( Tx_Buffer, countof(Tx_Buffer)-1);
```

除此之外，配置好以下两个步骤，STM32 的串口还支持 printf 函数：

1. 驱动程序里包含以下代码，这段引入 printf 函数支持的代码在 usart.c 文件的最上方和最下方，这段代码加入之后便可以通过 printf 函数向串口发送我们需要的内容，方便开发过程中查看代码执行情况以及一些变量值。这段代码不需要修改，主函数包括头文件 uart.h 即可：

```c
/* Private function prototypes -----------------------------------------------*/
#ifdef __GNUC__
  /* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small
     set to 'Yes') calls __io_putchar() */
  #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
  #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */
```

```
PUTCHAR_PROTOTYPE
{
/* 将Printf内容发往串口 */
  USART_SendData(USART1,(uint8_t)  ch);
  while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET)
  {}

  return (ch);
}
```

2.工程要使用 microlib 库，它是缺省 C 库的备选库。 它旨在与需要装入到极少量内存中的深层嵌入式应用程序配合使用。如图 3-3 在工程的配置里勾选：

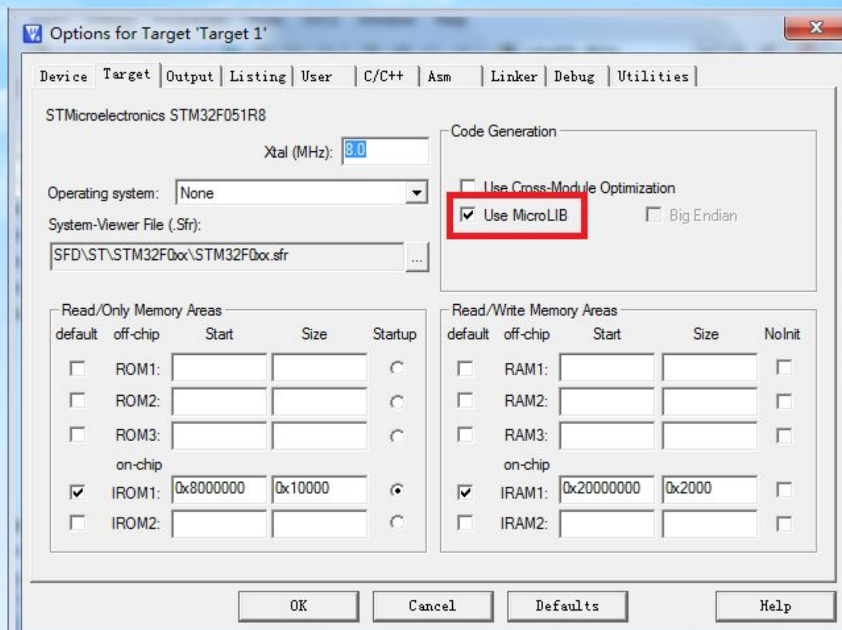


图 3- 3 配置工程

配置好以上两点之后，我们就可以调用 printf 函数了。

```
printf("welcome to use microcloud product!");
```

除了主函数的代码之外，我们还在中断文件里有一段重要的代码：

```
void USART1_IRQHandler(void)
{
    uint8_t buf[1]={0x00};  // 定义接收数据变量数组

  if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
  {
    buf[0] = USART1->RDR;   // 从RXFIFO中读取接收到的数据
    UART_send_byte(buf[0]);
  }
  return;
}
```

对于 STM32 来说，因为我们开启了接收中断，所以当外界有数据发送进来的时候，便会触发这个中断，从而执行这个函数，此函数的作用就是将接收到的数据再原封不动的发送出去。

至此，我们将这个工程的重要部分讲解完毕，接下来我们看一下现象。

6.5程序下载与测试

本工程实现的是：单片机自身不断发送“good welcome to use microcloud product!”。同时一旦电脑给开发板发送字符串，字符串即可在串口调试助手的接收窗口显示出来。

首先参照串口下载章节，将串口连接好，将程序下载到开发板。

然后打开串口调试助手，选择好 com 口之后，波特率设置为 115200，点击 open，就可以看到接收窗口有数据产生了（如图 3- 4）：

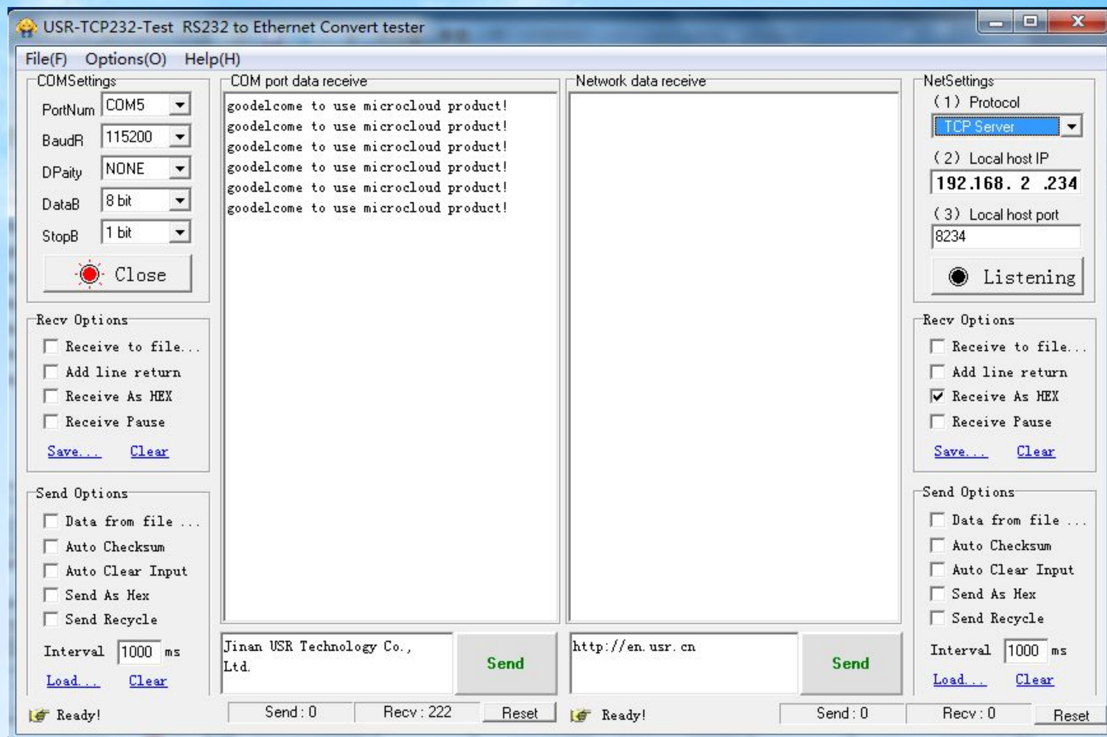


图 3- 4 串口显示

点击下面的 Send 按钮，可以看到图 3- 5，接收串口也会接收回数据的：

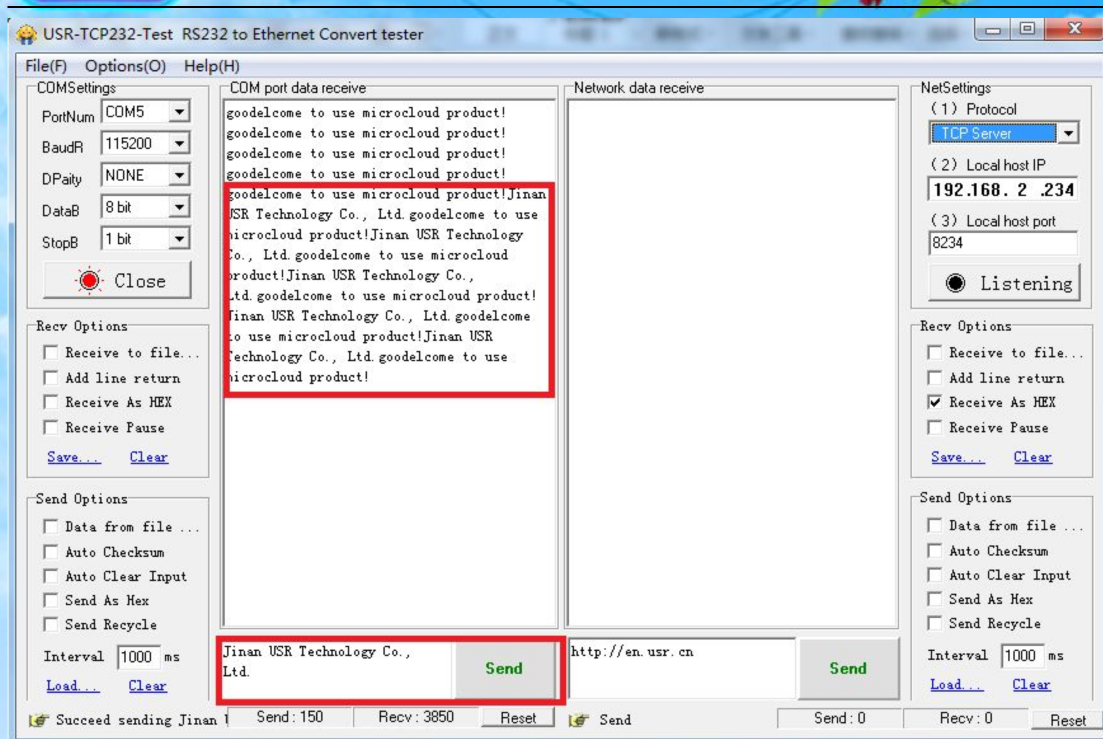


图 3- 5 串口收数

至此我们就将收发测试的部分讲解完毕了。

第 7 章 温湿度传感器读取

此部分我们通过 GPIO 口来读取温湿度传感器的值，然后通过串口将数据发送出去。

7.1温湿度传感器 DHT11 简介

DHT11 数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器。它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性与卓越的长期稳定性。

1、DHT11 的典型应用电路如图 4- 1 所示：

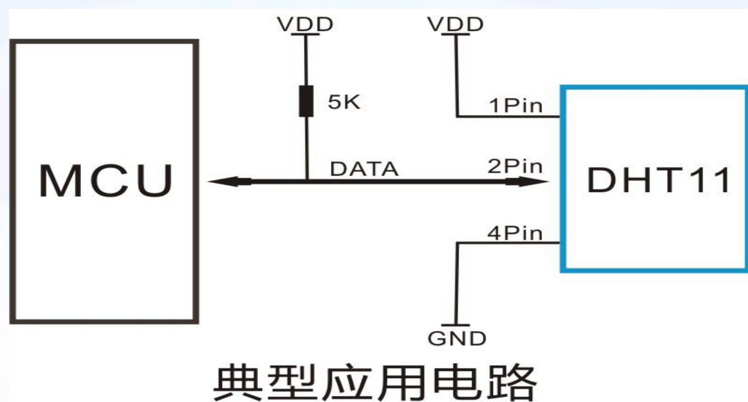


图 4- 1DHT11 的典型应用电路

2、DHT11 的引脚功能如表 4- 1 所示：

表 4- 1DHT11 的引脚功能

Pin	名称	注释
1	VDD	供电 3－5.5VDC
2	DATA	串行数据，单总线
3	NC	空脚，请悬空
4	GND	接地，电源负极

3、DHT11采用的是单线双向的串行接口

DATA 引脚用于微处理器与 DHT11之间的通讯和同步,采用单总线数据格式,一次通讯时间4ms左右,数据分小数部分和整数部分,具体格式在下面说明,当前小数部分用于以后扩展,现读出为零.操作流程如下：

一次完整的数据传输为40bit,高位在前。

数据格式:8bit湿度整数数据+8bit湿度小数数据
+8bi温度整数数据+8bit温度小数数据
+8bit校验和

数据传送正确时校验和数据等于**“8bit湿度整数数据+8bit湿度小数数据+8bit温度整数数据+8bit温度小数数据”**所得结果的末8位。

用户MCU发送一次开始信号后,DHT11从低功耗模式转换到高速模式,等待主机开始信号结束后,DHT11发送响应信号,送出40bit的数据,并触发一次信号采集,用户可选择读取部分数据.从模式下,DHT11接收到开始信号触发一次温湿度采集,如果没有接收到主机发送开始信号,DHT11不会主动进行温湿度采集.采集数据后转换到低速模式。通讯过程如图4- 2所示

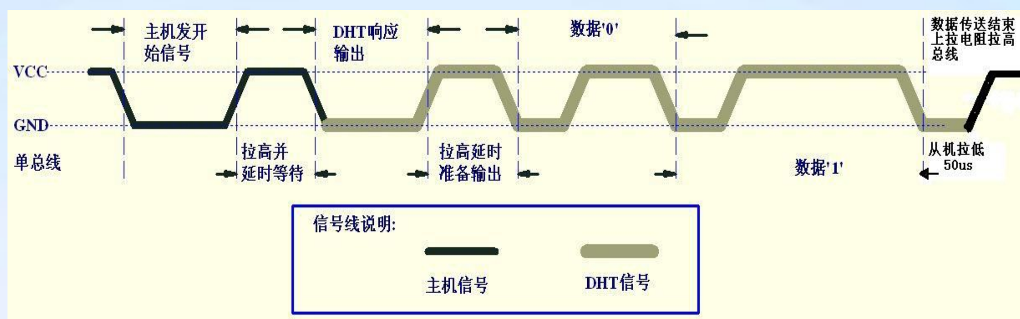


图 4- 2DHT11 时序图 1

总线空闲状态为高电平,主机把总线拉低等待DHT11响应,主机把总线拉低必须大于18毫秒,保证DHT11能检测到起始信号。DHT11接收到主机的开始信号后,等待主机开始信号结束,然后发送80us低电平响应信号.主机发送开始信号结束后,延时等待20-40us后, 读取DHT11的响应信号,主机发送开始信号后,可以切换到输入模式,或者输出高电平均可, 总线由上拉电阻拉高。如图4- 3所示：

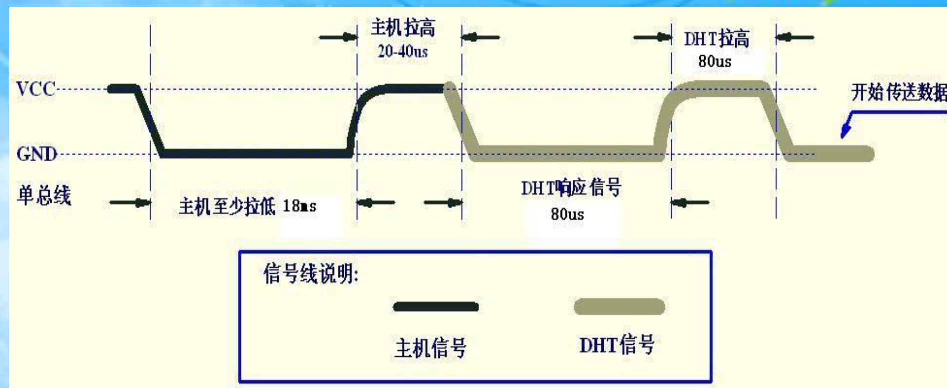


图 4- 3DHT11 时序图 2

总线为低电平,说明DHT11发送响应信号,DHT11发送响应信号后,再把总线拉高80us,准备发送数据,每一bit数据都以50us低电平时隙开始,高电平的长短定了数据位是0还是1.格式见下面图示.如果读取响应信号为高电平,则DHT11没有响应,请检查线路是否连接正常.当最后一bit数据传送完毕后，DHT11拉低总线50us,随后总线由上拉电阻拉高进入空闲状态。

数字0信号表示方法如图4- 4所示：

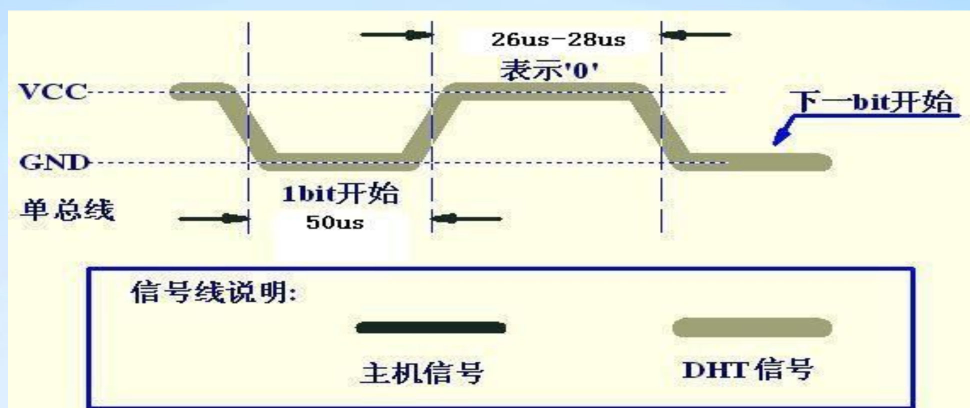


图 4- 4DHT11 时序图 3

数字1信号表示方法.如图4- 5所示：

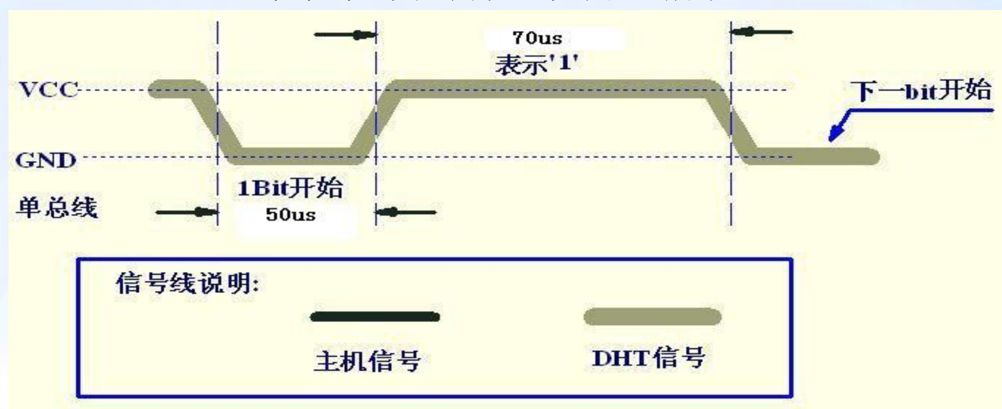


图 4- 5DHT11 时序图

对于串口的知识请参考上一章内容，这里就不多赘述了。

7.2硬件电路原理

DHT11 硬件原理图如图 4- 6 所示：

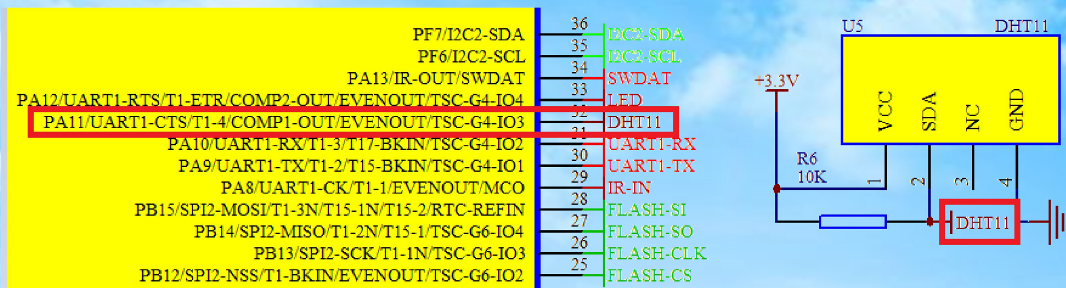


图 4- 6DHT11 硬件原理图

由上图可以看出，DHT11 是单线总线协议，只需要一个 GPIO 口 PA11 就可以驱动了。

7.3软件程序与注解

双击打开我们资料包里温湿度采集的例程（路径为..\STM32+linux 资料包\例程\lcd\user\DHT11.Uv2），可以看到如图 4- 7 的工程：

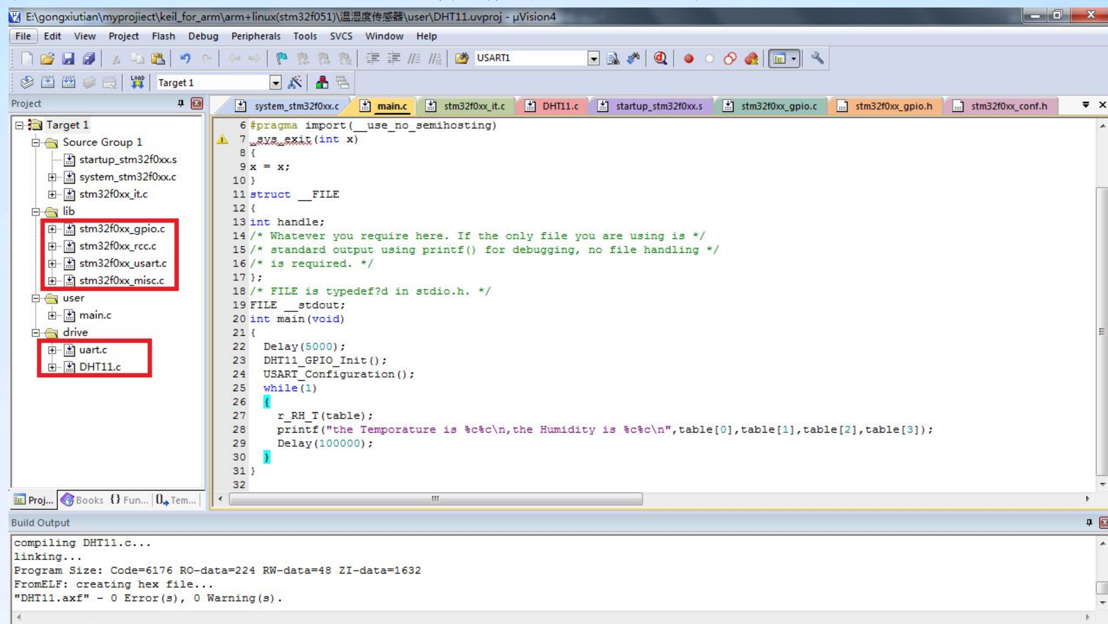


图 4- 7 温湿度采集的例程

虽然工程里包括了中断的文件，但是这里我们没有用到，所以不需要太关心，这里我们主要看主函数 main.c 和温湿度传感器驱动 DHT11.c，先看主函数，它包括的头文件有：

```
#include "stm32f0xx.h"
#include "DHT11.h"
#include "uart.h"
```

然后一段支持 printf 函数的代码，内容可以不必了解，感兴趣的同学可以上网查一查：

```
//以下代码是对printf函数的支持代码，加入此代码可以不用勾选微型库
#pragma import(__use_no_semihosting)
_sys_exit(int x)
{
x = x;
}
struct __FILE
{
int handle;
/* Whatever you require here. If the only file you are using is */
/* standard output using printf() for debugging, no file handling */
/* is required. */
};
/* FILE is typedef?d in stdio.h. */
FILE __stdout;
```

然后是主函数：

```
int main(void)
{
  Delay(5000);
  DHT11_GPIO_Init();
  USART_Configuration();
  while(1)
  {
    r_RH_T(table);
    printf("the Temporature is %c%c\n,the Humidity is %c%c\n",table[0],table[1],table[2],table[3]);
    Delay(100000);
  }
}
```

主函数先是一段时间的延时，这个可有可无，然后是对 DHT11 和串口的初始化，在主循环里 r_RH_T(table);将温湿度的数值读取到 table 数组里，然后 printf 打印到串口。接下来我们看一下 DHT11 的初始化函数和读取数据函数：

1）DHT11 的初始化函数 DHT11_GPIO_Init();

```
void DHT11_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
  GPIO_InitStruct.GPIO_Pin = DHT_PIN  ;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Speed =GPIO_Speed_Level_3;
//  GPIO_InitStruct.GPIO_PuPd =GPIO_PuPd_DOWN;
  GPIO_Init(DHT_PORT, &GPIO_InitStruct);
  GPIO_SetBits(DHT_PORT, DHT_PIN  );
}
```

DHT11 的初始化实际上就是将 PA11 口设置成推挽输出模式，同时要使能 GPIOA 的时钟。

2）DHT11 读取数据函数 r_RH_T();

```
void r_RH_T(unsigned char *data)
{
  unsigned char U8temp;
  unsigned char  U8T_data_H_temp,U8T_data_L_temp,U8RH_data_H_temp,U8RH_data_L_temp,U8checkdata_temp;
  SDA_OUT;
  GPIO_ResetBits(DHT_PORT, DHT_PIN);
  Delay(120);
  GPIO_SetBits(DHT_PORT, DHT_PIN);
  SDA_IN;
  {GPIOB->MODER&=0XFF3FFFFF;GPIOB->MODER|=0<<22;}
  Delay_10us();Delay_10us();
  Delay_10us();Delay_10us();
  if(GPIO_ReadInputDataBit(DHT_PORT, DHT_PIN)==0)     //T !
  {
    while(GPIO_ReadInputDataBit(DHT_PORT, DHT_PIN)==0);
    while(GPIO_ReadInputDataBit(DHT_PORT, DHT_PIN)==1);
    U8RH_data_H_temp=COM();
    U8RH_data_L_temp=COM();
    U8T_data_H_temp=COM();
    U8T_data_L_temp=COM();
    U8checkdata_temp=COM();
    SDA_OUT;
    GPIO_SetBits(DHT_PORT, DHT_PIN);
    U8temp=(U8T_data_H_temp+U8T_data_L_temp+U8RH_data_H_temp+U8RH_data_L_temp);
    if(U8temp==U8checkdata_temp)
    {
      *data=U8RH_data_H_temp/10+0x30;
      *(data+1)=U8RH_data_H_temp%10+0x30;
      *(data+2)=U8T_data_H_temp/10+0x30;
      *(data+3)=U8T_data_H_temp%10+0x30;
    }
  }
}
```

函数定义了几个变量用来接收温度，湿度以及校验的数据，先是将数据口设置成输出模式（SDA_OUT），然后将 PA11 拉低约 18ms 使器件复位，然后拉高，将 IO 口设置成输入模式（SDA_IN），复位以后等待约 40s 之后，器件会回应一个低电平，所以有一个 if 的语句，如果拉低了，那么就跳过这段低电平：

while(GPIO_ReadInputDataBit(DHT_PORT, DHT_PIN)==0);

while(GPIO_ReadInputDataBit(DHT_PORT, DHT_PIN)==1);

这时候开始连续读取五个字节的数据，分别是湿度整数部分、湿度小数部分、温度整数部分、温度小数部分和校验数据(U8RH_data_H_temp=COM();等)。读取完毕之后将总线重新设置成输出模式（SDA_OUT），然后计算校验，如果校验正确，那就将数据转化为字符型，存到数组 data 里，温湿度的小数部分属于保留位，所以没有进行转化，直接忽略掉了。读取单个字节的函数 COM()如下图所示：

```
unsigned char COM(void)
{
  unsigned char i,U8FLAG,U8temp,U8comdata;
  for(i=0;i<8;i++)
  {
      U8FLAG=2;
      while((!GPIO_ReadInputDataBit(DHT_PORT, DHT_PIN))&&U8FLAG++);
      Delay_10us();
      Delay_10us();
      Delay_10us();
      U8temp=0;
      if(GPIO_ReadInputDataBit(DHT_PORT, DHT_PIN)==1)
        U8temp=1;
      U8FLAG=2;
      while((GPIO_ReadInputDataBit(DHT_PORT, DHT_PIN))&&U8FLAG++);

      if(U8FLAG==1)break;

      U8comdata<<=1;
      U8comdata|=U8temp;
  }
  return U8comdata;
}
```

关于函数里的几个宏定义（SDA_OUT、SDA_IN）可以在 DHT11.h 里看到：

```
#define DHT_PIN       GPIO_Pin_11
#define DHT_PORT       GPIOA
#define SDA_OUT {GPIOA->MODER&=0XFF3FFFFF;GPIOA->MODER|=1<<22;}
#define SDA_IN {GPIOA->MODER&=0XFF3FFFFF;GPIOA->MODER|=0<<22;}
#define SDA_HIGH GPIO_SetBits(DHT_PORT, DHT_PIN)
#define SDA_LOW GPIO_ResetBits(DHT_PORT, DHT_PIN)
#define SDA_READ GPIO_ReadInputDataBit(DHT_PORT, DHT_PIN)
```

至此，我们将这个工程的重要部分讲解完毕，接下来我们看一下现象。

7.4程序下载与测试

此部分我们通过 GPIO 口来读取温湿度传感器的值，然后通过串口将数据发送出去。

插上串口，将程序下载到开发板（参考串口下载），打开串口调试助手，选择好 com 口之后，波特率设置为 115200，点击 open，就可以看到接收窗口有数据产生了：

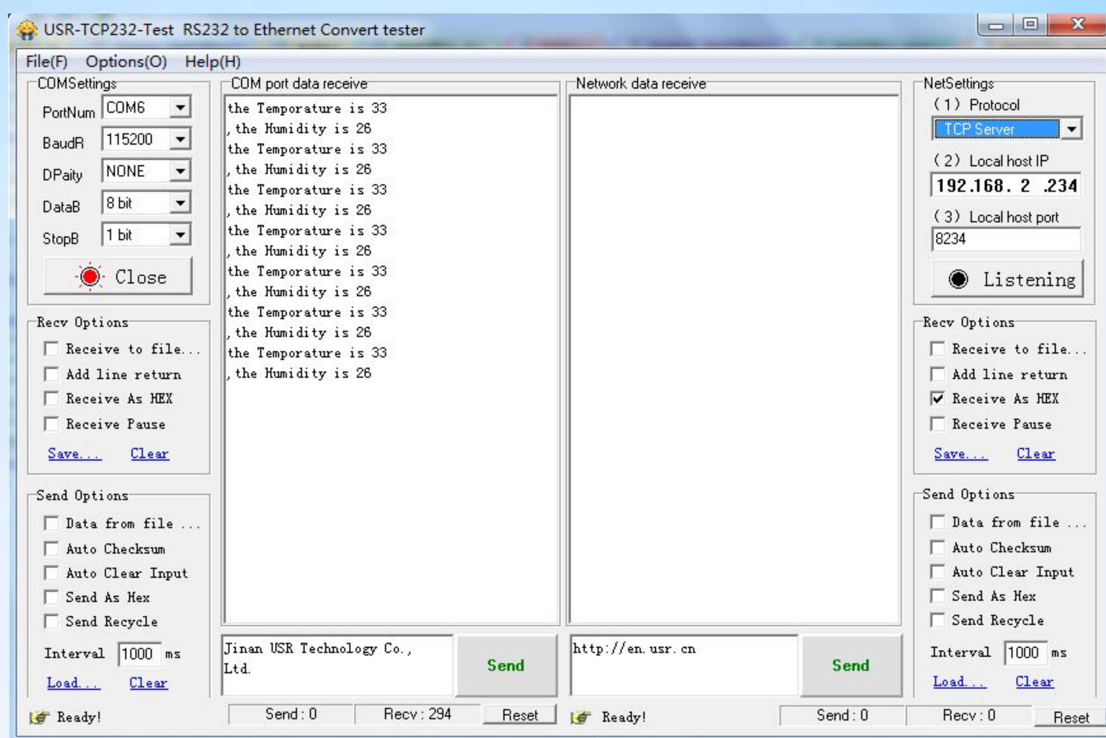


图 4- 8 显示结果

第 8 章 LCD 液晶屏驱动与显示

此部分，我们通过通用 IO 口来驱动液晶屏，如何使用 GPIO 在之前已经介绍过，这里不再详述，这里我们着重讲解一下关于液晶屏的知识。

8.1R61580IC 液晶模块简介

液晶显示器(LCD)英文全称为 Liquid Crystal Display，它一种是采用了液晶控制透光度技术来实现色彩的显示器。本学习板的配套 LCD 是 2.4 寸 TFT LCD，由 240*320 个像素点组成显像的，每个独立的像素色彩是由红、绿、蓝(R、G、B)三种基本色来控制。此 LCD 由 R61580 驱动，每个像素点对应一个 RAM，而 RAM 中的数据是 STM32 单片机通过八位通用 IO 口分两次并行写入的，RAM 中的数据决定了每个点的颜色，该数据为 18 位（R、G、B 各 6 位控制），该数据接口可以由 18 位接口，16 位接口，8 位接口或者是 9 位接口控制，这里我们才有 8 位接口分两次写入数据，即每个像素点由 16 位数据写入 18 个位。如何用 16 位数据写入 18 个位呢？因为每种颜色由 6 位控制，最后一个位是 0 还是 1，从视觉上看来对整体颜色效果是几乎没有差别的，所以就将红色的最高位和红色的最低位由同一个数据位控制，将蓝色的最高位和最低位由同一个数据位控制，如图所示，从而实现 16 位数据控制 18 位数据的颜色啦！

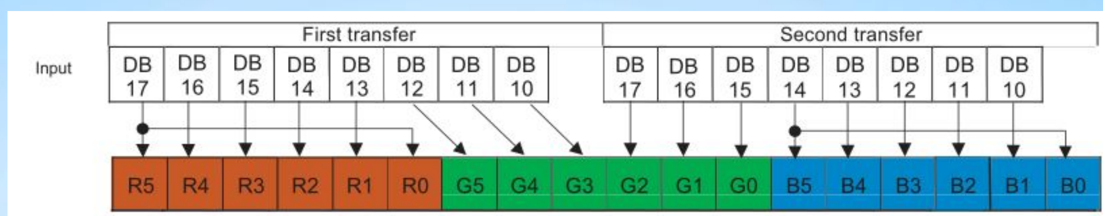


图 5- 1GBR

纯红色所对应的 16 进制数为 0xf8，0x00（11111000,00000000）；纯绿色所对应的 16 进制数为 0x07，0xe8；纯蓝色所对应的 16 进制数为 0x00，0x1f；显然红色有 2^5=32 种表现度，绿色有 2^6=64 种表现度，蓝色有 2^5=32 种表现度，可见，每个独立的像素就可以呈现出 32×64×32=65536 种色彩。因此通过控制单片机发送的数据就可以使液晶屏呈现五彩斑斓的图片啦！

管脚功能描述如图 5- 2 所示：

1~4	DB0~DB3	I/O Host processor	*SEE THE **Description of** 23~30PIN*	VCC1 OR GND
5	GND1	Power supply	POWER GROUND	-
6	VCC1	Power supply	POWER 1.8V/2.8V	-
7	/CS	I Host processor	Chip select signal. Amplitude: VCC1-GND Low: the driver is selected and accessible High: the driver is not selected and not accessible.	VCC1
8	RS	I Host processor	Register select signal.Amplitude: VCC1-GND Low: select Index register High: select control register	GND
9	/WR	I Host processor	Write strobe signal in 80-system bus interface operation and enables write operation when WR is low. Amplitude: VCC1-GND	-
10	/RD	I Host processor	Read strobe signal in 80-system bus interface operation and enables read operation when RDX is low. Amplitude: VCC1-GND	GND
11	IM0	I Host processor	IM0 INTERFACE MODE DB PIN 0 80system 16 bit interface DB15~0 1 80system 8 bit interface DB15~8	-
12	NC (X+)	-	reserved	-
13	NC(Y+)	-	reserved	-
14	NC(X-)	-	reserved	-
15	NC(Y-)	-	reserved	-
16	LED-A	LED driver	LED ANODE	open
17	LED-1	LED driver	LED 1(CATHODE)	open
18	LED-2	LED driver	LED 2(CATHODE)	open
19	LED-3	LED driver	LED 3(CATHODE)	open
20	LED-4	LED driver	LED 4(CATHODE)	open
21	NC	-	reserved	-
22	DB4	I/O Host processor	*SEE the **Description of** 23~30PIN*	VCC1 OR GND
23~30	DB8~ DB15	I/O Host processor	16-bit parallel bi-directional data bus for MPU system interface mode Serves as an input data bus for MPU I/F. 8-bit I/F: DB[15:8] is used. 16-bit I/F: DB[15:0] is used.	VCC1 OR GND
31	/RESET	I Host processor	Reset pad. Initializes the IC when it is low. Must be reset after power-on.	-
32	VCI	Power supply	POWER 2.8V(TYP)	-
33	VCC2	Power supply	POWER 2.8V(TYP)	-
34	GND	Power supply	POWER GROUND	-
35~37	DB5~ DB7	I/O Host processor	*SEE the **Description of** 23~30PIN*	VCC1 OR GND

图 5- 2 管脚功能描述

上面标红框的是软件需要配置的引脚，其他引脚在硬件上已经连接好，我们不用太关心，下面我们介绍一下需要软件控制的引脚的功能。

7 脚 CS：

该引脚是 LCD 片选引脚,低电平有效，当 CS 为低电平时，允许对驱动器读写操作，CS 为高时，不允许对驱动器操作。

8 脚 RS ：

该引脚是显示数据/指令选择输入寄存器选择引脚。当该位置“0”时，单片机向 LCD 写入的是命令，反之，当该位置“1”时，写入的是 LCD 的显示数据。

9 脚 WR：

该脚是脉冲选通信号，每当此引脚变为低电平的时候，并行数据口上的数据被允许写入驱动器。

10 脚 RD：

此引脚为读或者写驱动器的信号，当该位置“0”时，单片机向 LCD 读取的数据或命令，反之，当该位置“1”时，是向 LCD 的写入数据或命令。在硬件上该引脚已被接为 VCC，也就是只能被写入，而不能被读取了。

11 脚 IM0：

此引脚为接口模式选择位，当该位为“0”时是 16 位数据操作，为“1”时是 8 位数据操作，在硬件上我们已将此位接为 VCC，即只能进行 8 位操作了。其它引脚按照表中所示，分别接电源(3.3v)或地(GND)。图 5- 3 是对驱动器读写的时序图：

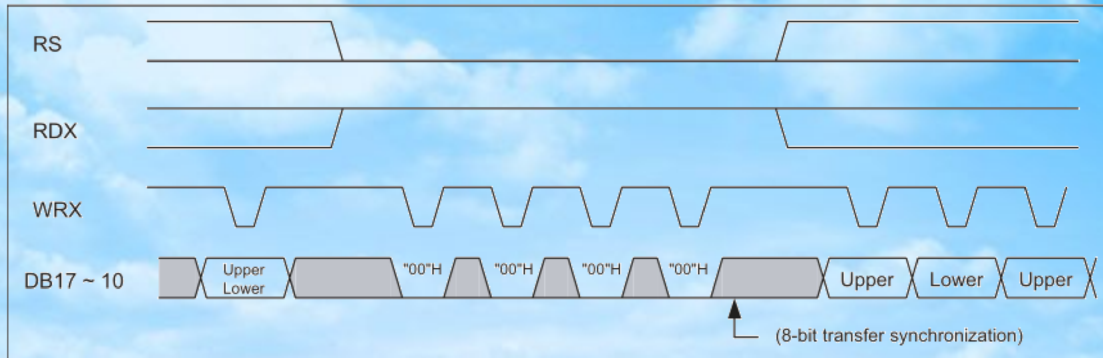


图 5- 3 读写的时序图

介绍完了 LCD 的引脚和时序，下面我们来介绍一下 R61580 的各种命令。

1、基本图像显示控制

驱动器输出控制（R60h）

基本图像显示控制（R61H）

垂直滚动控制（R6Ah）

这几个寄存器各个位的功能如图 5- 4 所示：

	R/W	RS	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0
R60	W	1	GS	0	NL[5]	NL[4]	NL[3]	NL[2]	NL[1]	NL[0]	0	0	SCN[5]	SCN[4]	SCN[3]	SCN[2]	SCN[1]	SCN[0]
	Default		0	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0
R61	W	1	0	0	0	0	0	0	0	0	0	0	0	0	0	NDL	VLE	REV
	Default		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R6A	W	1	0	0	0	0	0	0	0	VL[8]	VL[7]	VL[6]	VL[5]	VL[4]	VL[3]	VL[2]	VL[1]	VL[0]
	Default		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图 5- 4 寄存器各个位的功能

NL[5:0]：设置用于驱动 LCD 时的 8 行间隔的行数。帧存储器地址映射是不受由 NL[5:0]设定的行数影响。设置的行数必须是相同或比所必需的液晶面板的尺寸的行数更多。这里通常设置为默认的值 320 行。

GS：设置扫描通过栅极驱动器的方向。设置与 SM 和 SS 位的组合 GS 位方便的显示模块的配置和显示方向。详细情况请看。。。

REV：通过设置 REV= 1 使图像的灰度级反转。这使得相同的一组数据 R61580 在液晶面板上是否是正常黑色或显示相同的图像白色。通常我们设置为 1。

VLE：垂直滚动显示使能位。当 VLE= 1 时，R61580 开始从由 VL[8:0]位决定的行数来显示基本的图像。垂直滚动是无法在外接显示器接口操作。在这种情况下，请务必设置 VLE=“0”。因此我们通常将这一位设置成 0。

NDL：设置在非点亮显示区域的源输出电平。NDL 位可以保持点亮的非显示区域。

VL[8:0]：设置基本图像的滚动量。基本图像以垂直方向滚动和从由 VL[8:0]确定



的行进行显示。确保 VL[8:0]≤320。

SCN[5:0]：定义栅极驱动器开始扫描的行位。

2、局部显示控制

局部图像显示位置（R80h）

局部图像帧存储器地址（起始行地址）（R81h）

局部图像帧存储器地址（终止行地址）（R82h）

这几个寄存器各个位的功能如图 5- 5 所示：

	R/W	RS	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0
R 80	W	1	0	0	0	0	0	0	0	PTDP [8]	PTDP [7]	PTDP [6]	PTDP [5]	PTDP [4]	PTDP [3]	PTDP [2]	PTDP [1]	PTDP [0]
	Default value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R 81	W	1	0	0	0	0	0	0	0	PTSA [8]	PTSA [7]	PTSA [6]	PTSA [5]	PTSA [4]	PTSA [3]	PTSA [2]	PTSA [1]	PTSA [0]
	Default value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R 82	W	1	0	0	0	0	0	0	0	PTE A[8]	PTE A[7]	PTE A[6]	PTE A[5]	PTE A[4]	PTE A[3]	PTE A[2]	PTE A[1]	PTE A[0]
	Default value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图 5- 5 寄存器各个位的功能

PTDP[8:0]：设置局部图像的显示位置。如果 PTDP0="9'h000"，局部图像是从基本图像的第一行中显示。

PTSA[8:0]，PTEA[8:0]：分别为局部图像设置帧存储器区域的起始行和结束行的地址。在设置时，请确保 PTSA≤PTEA。

3、显示控制寄存器 1(07H)

R/W	RS	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0
W	1	0	0	0	PTDE	0	0	0	BASE E	0	0	0	0	COL	0	0	0
Default value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

COL：当 COL=1，灰度级放大器显示图像等，使功耗减少。此外，只有 8 种颜色显示。

Table 16

COL	Display color
0	262,144
1	8

BASEE：基本图像显示使能位。

BASEE=0：无基本图像显示。该 R61580 驱动液晶与非点亮的显示级别或驱动器只有部分图像显示区域。

BASEE=1：基本的图像显示在面板上。

PTDE：PTDE 是显示器使局部图像的位。

PTDE=0：部分图像不显示。只有基础的图像显示。

PTDE=1：显示部分图像。写 BASEE=0 来关闭一个基本图像。

Table 17

BASEE	PTDE	VLE	COL	State
0	0	*	*	Halt display operation
1	0	0	0	262,144-color display operation
1	0	0	1	8-color display operation
1	0	1	0	262,144-color display operation with scroll function enabled
0	1	*	0	262,144-color partial display operation
0	1	*	1	8-color partial display operation

4、显示控制寄存器 2(08H)

R/W	RS	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0
W	1	FP [7]	FP [6]	FP [5]	FP [4]	FP [3]	FP [2]	FP [1]	FP [0]	BP [7]	BP [6]	BP [5]	BP [4]	BP [3]	BP [2]	BP [1]	BP [0]
Default value		0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0

FP[7:0]：设置前沿周期（结束后的空白期）的行数。
BP[7:0]：设置后沿周期（显示前做了一个空白期）的行数。
关于此寄存器的详细情况请参看 R61580ic 的规格书。
5、显示控制寄存器 3(09H)

R/W	RS	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0
W	1	0	0	0	0	0	PTS [2]	PTS [1]	PTS [0]	0	0	PTG	0	ISC [3]	ISC [2]	ISC [1]	ISC [0]
Default value		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

SC [3:0]：当 PTG [1:0]在非显示区域的驱动周期选择间隔扫描时，设置扫描周期。该扫描周期为 n 的帧周期，其中 n 是一个 3 至 31 的奇数。
PTG：设置在非显示区域的扫描模式。
PTS [2:0]：设置在非显示区域内的驱动源输出电平。
6、显示屏地址控制
横向帧存储器地址（起始地址）（R50h）
横向帧存储器地址（结束地址）（R51h）
垂直帧存储器地址（起始地址）（R52h）
垂直帧存储器地址（结束地址）（R53h）

	R/W	RS	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0
R 50	W	1	0	0	0	0	0	0	0	0	HSA [7]	HSA [6]	HSA [5]	HSA [4]	HSA [3]	HSA [2]	HSA [1]	HSA [0]
	Default		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R 51	W	1	0	0	0	0	0	0	0	0	HEA [7]	HEA [6]	HEA [5]	HEA [4]	HEA [3]	HEA [2]	HEA [1]	HEA [0]
	Default		0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1
R 52	W	1	0	0	0	0	0	0	0	VSA [8]	VSA [7]	VSA [6]	VSA [5]	VSA [4]	VSA [3]	VSA [2]	VSA [1]	VSA [0]
	Default		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R 53	W	1	0	0	0	0	0	0	0	VEA [8]	VEA [7]	VEA [6]	VEA [5]	VEA [4]	VEA [3]	VEA [2]	VEA [1]	VEA [0]
	Default		0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1

HSA[7:0]，HEA[7:0]：HSA[7:0]和 HEA[7:0]分别是窗口地址区在水平方向上的

起始地址和结束地址。HSA[7:0]和 HEA[7:0]指定了写入数据的水平范围。要在对帧存储器的写操作之前设置 HSA[7:0]和 HEA[7:0]的值。

VSA[8:0]，VEA[8:0]：VSA[8:0]和 VEA[8:0]分别是窗口地址区在垂直方向上的起始地址和结束地址。VSA[8:0]和 VEA[8:0]指定了写入数据的垂直范围。要在对帧存储器的写操作之前设置 VSA[8:0]和 VEA[8:0]的值。

以上这些命令为 R61580 初始化时需要根据自己的需要特殊设置的，当然还有很多其他的命令，这些命令一般只需要使用系统默认值就可以了，不需要特殊的设置，在这里就不一一赘述了，可以参看 R61580ic 的规格书进行详细的了解。

8.2硬件电路原理

Lcd 液晶屏与 STM32 单片机的硬件原理图如图 5- 6 所示：

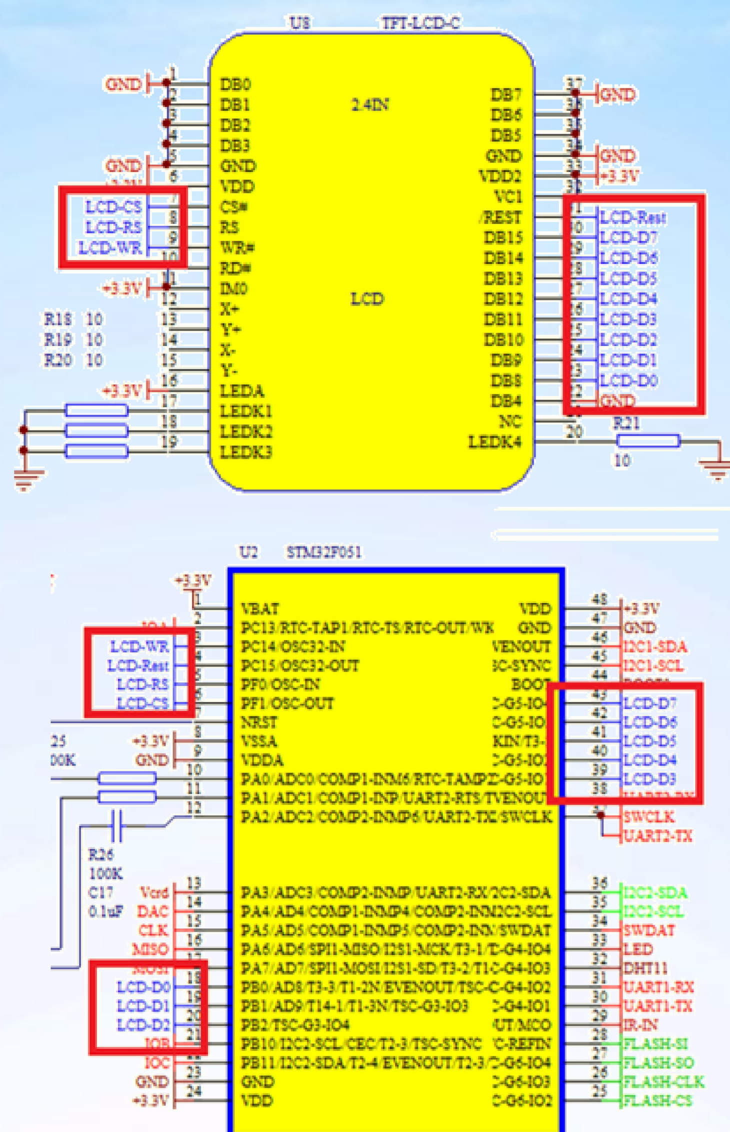


图 5- 6 硬件原理图

8.3软件程序与注解

双击打开我们资料包里液晶屏的例程（路径为..\STM32+linux 资料包\例程\lcd\user\lcd.Uv2），可以看到如图 5- 7 的工程：

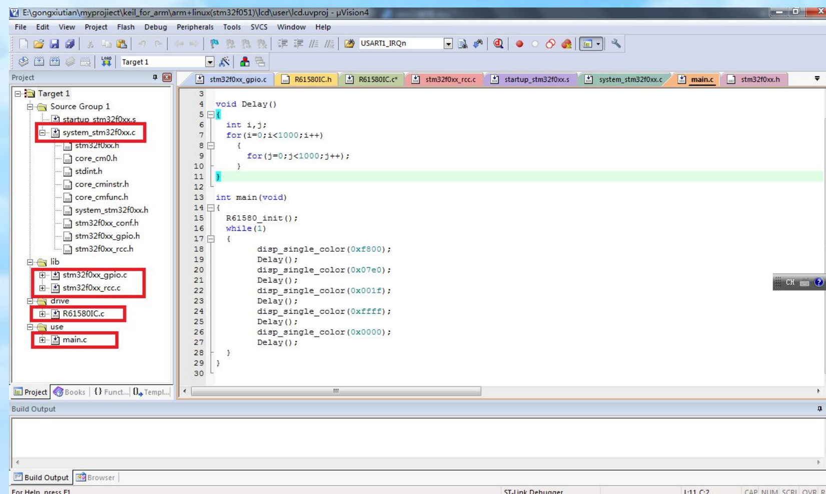


图 5- 7 液晶屏的例程

工程包含的组和 led 灯闪烁的工程的组是一样的。这里我们主要看主函数 main.c 和液晶驱动函数 R61580IC.c，先看主函数，它包括的头文件有：

```
#include "stm32f0xx.h"
#include "R61580IC.h"
```

然后一个简单的延时函数：

```
void Delay()
{
  int i,j;
  for(i=0;i<1000;i++)
    {
      for(j=0;j<1000;j++);
    }
}
```

然后是主函数：

```
int main(void)
{
  R61580_init();
  while(1)
  {
        disp_single_color(0xf800);
        Delay();
        disp_single_color(0x07e0);
        Delay();
        disp_single_color(0x001f);
        Delay();
        disp_single_color(0xffff);
        Delay();
        disp_single_color(0x0000);
        Delay();
  }
}
```

主函数里很简单，一共就三个函数，液晶屏的初始化 R61580_init()，显示单个屏

的颜色 disp_single_color()和延时 Delay()。

调到 R61580_init()函数(右击函数，Go To Definition of...)，可以看到如下代码：

```
void R61580_init(void)
{
  R61580_GPIO_Init();
  ResetR61580();
  R61580WriteComData(0x0000,0x0000);
  R61580WriteComData(0x0000,0x0000);
  R61580WriteComData(0x0000,0x0000);
  R61580WriteComData(0x0000,0x0000);
  R61580WriteComData(0x00A4,0x0001);
  delay(50);
  R61580WriteComData(0x0060,0xA700);
  R61580WriteComData(0x0008,0x0404);

  R61580WriteComData(0x0090,0x0019);//70Hz
  R61580WriteComData(0x0010,0x0530);//BT,AP  is default value
  R61580WriteComData(0x0011,0x0237);//DC1,DC0,VC  is default value
  R61580WriteComData(0x0012,0x01BC);//Turns power supply on  Sets the factor to genera
  R61580WriteComData(0x0013,0x1B00);  //Set VCOM alternating amplitude in the range of
  delay(50);

  R61580WriteComData(0x0001,0x0100);
  R61580WriteComData(0x0002,0x0200);
  R61580WriteComData(0x0003,0x1030);
  R61580WriteComData(0x0009,0x002F);
  R61580WriteComData(0x000A,0x0008);
  R61580WriteComData(0x000C,0x0000);
  R61580WriteComData(0x000D,0xD000);
  R61580WriteComData(0x000E,0x0030);
  R61580WriteComData(0x000F,0x0000);
  R61580WriteComData(0x0020,0x0000);//H Start
  R61580WriteComData(0x0021,0x0000);//V Start
  R61580WriteComData(0x0029,0x0061);
  R61580WriteComData(0x0050,0x0000);
  R61580WriteComData(0x0051,0xD0EF);
  R61580WriteComData(0x0052,0x0000);
  R61580WriteComData(0x0053,0x013F);
  R61580WriteComData(0x0061,0x0001);
  R61580WriteComData(0x006A,0x0000);
  R61580WriteComData(0x0080,0x0000);
  R61580WriteComData(0x0081,0x0000);
  R61580WriteComData(0x0082,0x005F);
  R61580WriteComData(0x0093,0x0701);

  R61580WriteComData(0x0007,0x0100);
//  WriteCom(0x0007);
//  WriteData(0x0173);
}
```

液晶屏初始化函数实际上就是对一大推的寄存器赋值，R61580ic 有几十个寄存器，这里我们就不一一介绍了，这里我们看一下如何从底层开始驱动液晶屏。

首先液晶屏是由 GPIO 驱动的，它包括 WR，CS，RST 以及并口数据传输 DATA 等引脚，为了方便编程以及对代码的阅读，对以上几个引脚进行了宏定义：

```
#include"stm32f0xx.h"
#define CS_PORT GPIOF
#define RS_PORT GPIOF
#define WR_PORT GPIOC
#define RST_PORT GPIOC              //Reset port
#define DATA_PORT GPIOB

#define CS_PIN GPIO_Pin_1
#define RS_PIN GPIO_Pin_0
#define WR_PIN GPIO_Pin_14
#define RST_PIN GPIO_Pin_15              //Reset pin
#define DATA_PIN (GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7)
```

R61580ic.c 里先要对这几个 GPIO 口进行初始化，原理不细说，请参看点亮



led 灯一章：

```
void R61580_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB|RCC_AHBPeriph_GPIOC|RCC_AHBPeriph_GPIOF, ENABLE);
  /***************set data_port as output mode******************/
  GPIO_InitStruct.GPIO_Pin = DATA_PIN  ;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Speed =GPIO_Speed_Level_3;
  GPIO_Init(DATA_PORT, &GPIO_InitStruct);
  GPIO_SetBits(DATA_PORT, DATA_PIN  );

  /***************set wr and rst as output mode******************/
  GPIO_InitStruct.GPIO_Pin = WR_PIN | RST_PIN ;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Speed =GPIO_Speed_Level_3;
  GPIO_Init(WR_PORT, &GPIO_InitStruct);
  GPIO_SetBits(WR_PORT, WR_PIN | RST_PIN  );
  /***************set cs and rs as output mode******************/
  GPIO_InitStruct.GPIO_Pin = CS_PIN | RS_PIN ;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Speed =GPIO_Speed_Level_3;
  GPIO_Init(CS_PORT, &GPIO_InitStruct);
  GPIO_SetBits(CS_PORT, CS_PIN | RS_PIN  );

}
```

然后是复位函数，就是将 RST 引脚拉低一段时间，再拉高，即可将 R61580ic 器件复位。

```
void ResetR61580(void)
{
  GPIO_ResetBits(RST_PORT, RST_PIN  );
  delay(1000);
  GPIO_SetBits(RST_PORT, RST_PIN );
  delay(500);

}
```

对于向器件写数据函数，先是片选拉低，然后 RS 引脚设置为高（写数据），然后将数据的高八位写到数据并口引脚，WR 引脚拉低拉高，将数据送入器件，再传送低八位，最后片选拉高，停止送数。

```
void WriteData(uint16_t data)
{
  GPIO_ResetBits(CS_PORT, CS_PIN  );      //cs=0
  GPIO_SetBits(RS_PORT, RS_PIN );         //rs=1
  GPIO_Write(DATA_PORT, /*(GPIO_ReadOutputData(DATA_PORT)&0xff00)|*/(data>>8));
  GPIO_ResetBits(WR_PORT, WR_PIN  );      //WR=0
  GPIO_SetBits(WR_PORT, WR_PIN );         //WR=1
  GPIO_Write(DATA_PORT, /*(GPIO_ReadOutputData(DATA_PORT)&0xff00)|*/(data&0x00ff)); //
  GPIO_ResetBits(WR_PORT, WR_PIN  );      //WR=0
  GPIO_SetBits(WR_PORT, WR_PIN );         //WR=1
  GPIO_SetBits(CS_PORT, CS_PIN  );      //cs=1
}
```

对于向器件写命令函数，和写数据函数相类似，只是 RS 引脚设置为低（写命令）。



```
void WriteCom(uint16_t com)
{
  GPIO_ResetBits(CS_PORT, CS_PIN  );       //cs=0
  GPIO_ResetBits(RS_PORT, RS_PIN );         //rs=0
  GPIO_Write(DATA_PORT,/*(GPIO_ReadOutputData(DATA_PORT)&0xff00)|*/(com>>8));
  GPIO_ResetBits(WR_PORT, WR_PIN  );       //WR=0
  GPIO_SetBits(WR_PORT, WR_PIN );          //WR=1
  GPIO_Write(DATA_PORT, /*(GPIO_ReadOutputData(DATA_PORT)&0xff00)|*/(com&0x00ff));  //
  GPIO_ResetBits(WR_PORT, WR_PIN  );       //WR=0
  GPIO_SetBits(WR_PORT, WR_PIN );         //WR=1
  GPIO_SetBits(CS_PORT, CS_PIN  );       //cs=1
}
```

以上两个函数在每次写数据或者是读数据的时候都要拉高片选，这样占用了不少的时间，液晶屏有 240*320 个数据点，这样若连续操作一个屏的话会很慢，因此我们还用直接操作寄存器的方法，写了一个不涉及片选的写单个数据的函数，以便可以加快写整个屏的速度：

```
void WriteWord(uint16_t data)
{

  GPIOB->ODR = data>>8;//GPIO_Write(DATA_PORT, /*(GPIO_ReadOutputData
  GPIOC->BRR = GPIO_Pin_14;//GPIO_ResetBits(WR_PORT, WR_PIN  );
  GPIOC->BSRR = GPIO_Pin_14;//GPIO_SetBits(WR_PORT, WR_PIN );
  GPIOB->ODR = data;//GPIO_Write(DATA_PORT, /*(GPIO_ReadOutputData(DA
  GPIOC->BRR = GPIO_Pin_14;       //WR=0
  GPIOC->BSRR = GPIO_Pin_14;        //WR=1


}
```

这个函数只是向器件的数据并口写两次数据。

```
void R61580WriteComData(uint16_t com,uint16_t val)
{
  WriteCom(com);
  WriteData(val);
}
```

此函数是向相应的寄存器里写数据的函数。

```
void set_window(unsigned int x0,unsigned int y0,unsigned int x1,unsigned int y1)
{
  R61580WriteComData(0x0050,x0);      // X Start Address Set
  R61580WriteComData(0x0051,x1);      // X End Address Set
  R61580WriteComData(0x0052,y0);      // Y Start Address Set
  R61580WriteComData(0x0053,y1);      // Y End Address Set
  WriteCom(0x0022);       // LCD_WriteCMD(GRAMWR);
}
```

设置要写数据的点的坐标，x0 是横坐标的起点，x1 是横坐标的终点，横坐标范围不超过 240；y0 是纵坐标的起点，y1 是纵坐标的终点，纵坐标范围不超过 320。

`WriteCom(0x0022);` 是准备向 RAM 写数据的命令，这个命令写完之后就可以向器件写 RGB 数据了。

```
void disp_single_color(uint16_t data)
{
  int i,j;
  set_window(0,0,240,320);
  GPIO_ResetBits(CS_PORT, CS_PIN  );       //cs=0
  GPIO_SetBits(RS_PORT, RS_PIN );         //rs=1
  for(i=0;i<320;i++)
  {
    for(j=0;j<240;j++)
    {
      WriteWord(data);
    }
  }
  GPIO_SetBits(CS_PORT, CS_PIN  );       //cs=1
}
```

这个函数是显示单个颜色图片的函数，先设置图片的坐标，片选拉低，然后向器件写入 240*320 个点的 RGB 数据，16 位数据，前 5bit 是红色，中间 6bit 是绿色，后 5bit 是蓝色，最后片选拉高。在主函数的循环里，显示不同的颜色，直接调用这个函数即可。

```
int main(void)
{
  R61580_init();
  while(1)
  {
        disp_single_color(0xf800);
        Delay();
        disp_single_color(0x07e0);
        Delay();
        disp_single_color(0x001f);
        Delay();
        disp_single_color(0xffff);
        Delay();
        disp_single_color(0x0000);
        Delay();
  }
}
```

至此，我们将这个工程的重要部分讲解完毕，接下来我们看一下现象。

8.4程序下载与测试

我们通过通用 IO 口来驱动液晶屏，从而实现液晶屏刷屏的功能。

将程序下载到开发板，就可以看到有不同的颜色在刷屏了，如图 5- 8 所示：

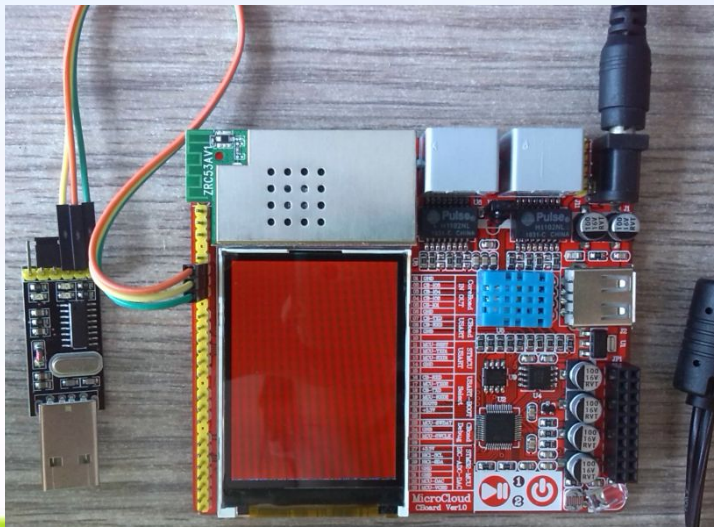


图 5- 8 刷屏显示红色

第 9 章 ADC 学习与光强采集

9.1ADC 原理简介

STM32 拥有 1~3 个 ADC（STM32F051 只有 1 个 ADC），这些 ADC 可以独立使用，也可以使用双重模式（提高采样率）。12 位 ADC 是一种逐次逼近型模拟数字转换器。它有多达 19 个通道，可测量 16 个外部和 3 个内部信号源。各通道的 A/D 转换可以单次、连续、扫描或间断模式执行。ADC 的结果可以左对齐或右对齐方式存储在 16 位数据寄存器中。 模拟看门狗特性允许应用程序检测输入电压是否超出用户定义的高/低阀值。

我们选择的 STM32F051R8 有 1 个 ADC。STM32 的 ADC 最大的转换速率为 1Mhz，也就是转换时间为 1us（在 ADCCLK=14M,采样周期为 1.5 个 ADC 时钟下得到），不要让 ADC 的时钟超过 14M，否则将导致结果准确度下降。我们本章仅介绍如何使用单次转换模式。

STM32 的 ADC 在单次转换模式下，只执行一次转换，该模式可以通过 软件用设置 ADSTART=1 启动 ADC 转换，也可以通过外部触发启动。

一旦所选择的通道转换完成，转换结果将被存在 ADC_DR 寄存器中，EOC（转换结束）标志将被置位，如果设置了 EOCIE，则会产生中断。然后 ADC 将停止，直到下次启动。

接下来，我们介绍一下我们执行单次转换需要用到的 ADC 寄存器。第一个要介绍的是 ADC 控制寄存器（ADC_CR）。ADC_CR 的各位描述：

12.12.3ADC 控制寄存器 (ADC_CR)

偏移地址：0x08

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AD CAL	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
rs															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	AD STP	Res.	AD START	AD DIS	AD EN
											rs		rs	rs	rs

这里我们不再详细介绍每个位，而是抽出几个我们本章要用到的位进行针对性的介绍，详细的说明及介绍，请参考《02_STM32F05x8 参考手册（中文)》第 12 章的相关章节。

ADSTART: ADC 开始转换命令。

该位由软件设置来启动 ADC 转换。一次转换可由立即启动 (由软件配置) 或硬件触发产生 (硬件触发配置) 两种方式来启动，启动方式由 EXTEN[1:0] 位的配置来决定。其位由硬件清零。

ADEN: ADC 使能命令

由软件设置该位来使能 ADC。一旦 ADRDY 标志置为 1 时表明 ADC 可供使用了。执行 ADDIS 命令后，ADC 关断且该位被硬件清零。

第二个需要介绍的是 ADC 配置寄存器，有两个配置寄存器 ADC_CFGR1 和



ADC_CFGR2，其中 ADC_CFGR2 这里我们没有用到，所以我们只介绍一下 ADC_CFGR1，

12.12.4 ADC 配置寄存器 1 (ADC_CFGR1)

偏移地址：0x0C

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	AWDCH[4:0]					Res.	Res.	AWD EN	AWD SGL	Res.	Res.	Res.	Res.	Res.	DISC EN
	rw	rw	rw	rw	rw			rw	rw						rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AUT OFF	AUT DLY	CONT	OVR MOD	EXTEN[1:0]		Res.	EXTSEL[2:0]			ALIGN	RES[1:0]		SCAN DIR	DMA CFG	DMA EN
rw	rw	rw	rw	rw			rw			rw	rw		rw	rw	rw

这个寄存器主要是对模拟看门狗，转换的模式，数据对齐的方式等进行配置。

CONT: 单次 / 连续转换模式

该位由软件设置和清除。 若该位置位，转换为连续模式直到该位清零。

0: 单次转换模式，1: 连续转换模式。因为我们是单次转换，所以这个位我们要设置为 0.

ALIGN: 数据对齐

该位由软件设置和清除用来选择数据的左或右对齐的方式。

RES[1:0]: 数据分辨率

这些位由软件改写，用于选择 ADC 转换的数据分辨率。

00: 12 位。01: 10 位。10: 8 位。11: 6 位。其数据对齐方式和分辨率的分布可如下图所示：

数据对齐与分辨率

ALIGN	RES	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0x0	0x0				DR[11:0]											
	0x1	0x00						DR[9:0]									
	0x2	0x00								DR[7:0]							
	0x3	0x00										DR[5:0]					
1	0x0	DR[11:0]												0x0			
	0x1	DR[9:0]										0x00					
	0x2	DR[7:0]								0x00							
	0x3	0x00								DR[5:0]						0x0	

SCANDIR: 扫描序列方向

该位由软件设置和清除来选择通道序列中的通道扫描方向。

0: 向前扫描 (从 CHSEL0 到 CHSEL16)

1: 向后扫描 (从 CHSEL16 到 CHSEL0)

这里我们只开启一个通道，扫描方向关系不大。

ADC_CFGR1 寄存器还有两个位是配置 DMA 的，这里我们没有用到，不多细述，

12.12.6 ADC 采样时间寄存器 (ADC_SMPR)

偏移地址：0x14

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SMP[2:0]		
													rw		

位 31:3　保留，必须保持为复位值 .

位 2:0　SMP[2:0]: 采样时间选择

这位用软件改写，用于选择所选通道的采样时间。

000: 1.5 ADC 时钟周期

001: 7.5 ADC 时钟周期

010: 13.5 ADC 时钟周期

011: 28.5 ADC 时钟周期

100: 41.5 ADC 时钟周期

101: 55.5 ADC 时钟周期

110: 71.5 ADC 时钟周期

111: 239.5 ADC 时钟周期

此寄存器用来确定 ADC 的采样时间，如上图所示。

12.12.8 ADC 通道选择寄存器 (ADC_CHSELR)

偏移地址：0x28

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CHSEL 17	CHSEL 16
														rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHSEL 15	CHSEL 14	CHSEL 13	CHSEL 12	CHSEL 11	CHSEL 10	CHSEL 9	CHSEL 8	CHSEL 7	CHSEL 6	CHSEL 5	CHSEL 4	CHSEL 3	CHSEL 2	CHSEL 1	CHSEL 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:18　保留，必须保持为复位值 .

位 17:0　CHSELx: 通道选择

这些位可由软件改写，用来定义所要转换序列的通道。

0: 输入通道 x 不被选为转换通道

1: 输入通道 x 被选为转换通道

ADC 一共有 19 个通道，ADC_CHSELR 可用来选通所需要转换的通道。

12.12.9 ADC 数据寄存器 (ADC_DR)

偏移地址：0x40

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

当 ADC 转换完毕之后可通过这个寄存器读出 ADC 转换后的数据。

最后一个要介绍的 ADC 寄存器为 ADC 中断和状态寄存器（ADC_ISR），该寄

存器保存了 ADC 转换时的各种状态。该寄存器的各位描述如图所示：

12.12.1ADC 中断和状态寄存器 (ADC_ISR)

偏移地址：0x00

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	AWD	Res.	Res.	OVR	EOS	EOC	EOSMP	ADRDY
								r_w1			r_w1	r_w1	rc_w1	r_w1	r_w1

这里我们要用到的是 EOC 位，

EOS: 序列转换结束标志

由 CHSEL 位所选的通道序列转换结束后，该位由硬件置位。其由软件对该位写 1 请零

0: 序列转换未完成 (或该事件标志由软件获取并清零)

1: 序列转换完成

我们通过判断该位来决定是否此次 AD 转换已经完成，如果完成我们就从 ADC_DR 中读取转换结果，否则等待转换完成。

通过以上寄存器的介绍，我们了解了 STM32 的单次转换模式下的相关设置，下面我们介绍使用库函数的函数来设定使用 ADC 的通道 3 进行 AD 转换。这里需要说明一下，使用到的库函数分布在 stm32f0xx_adc.c 文件和 stm32f0xx_adc.h 文件中。下面讲解其详细设置步骤：

1）开启 PA 口时钟和 ADC1 时钟，设置 PA1 为模拟输入。

STM32F051 的 ADC 通道 3 在 PA3 上，所以，我们先要使能 PORTA 的时钟和 ADC 时钟，然后设置 PA3 为模拟输入。使能 GPIOA 用 RCC_APBPeriphClockCmd 函数，使能 ADC 用 RCC_APB2PeriphClockCmd 函数，设置 PA3 的输入方式，使用 GPIO_Init 函数即可。

2）复位 ADC，开启 ADC 时钟之后，我们要复位 ADC，将 ADC 的全部寄存器重设为缺省值。ADC 时钟可从专门的 14 MHz RC 振荡器 (HSI14) 或 PCLK /2(或 /4) 得到。当 ADC 时钟源于 PCLK 时，其 ADC 时钟为 PCLK 时钟的反相信号。14MHzr 的 HSI RC 振荡器可以软件配置成由 ADC 接口控制的打开 / 关闭 (自动关) 模式或者常开模式。默认开启的时钟是内部 HSI14M 的时钟。

ADC 时钟复位的方法是：

```
ADC_DeInit(ADC1);
```

这个函数非常容易理解，就是复位指定的 ADC。

3）初始化 ADC1 参数，设置 ADC1 的工作模式等相关信息。

在复位 ADC 之后，我们就可以开始 ADC 的模式配置了，设置单次转换模式、分辨率、数据对齐方式等都在这一步实现。我们这里只有一个通道，并且是单次转换的，所以设置通道数为 1。这些在库函数中是通过函数 ADC_Init 实现的，下面我们看看其定义：

```
void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)
```

第一个参数是指定 ADC 。第二个参数跟其他外设初始化一样，同样是通过设置结构体成员变量的值来设定参数。

```
typedef struct
{
  uint32_t ADC_Resolution;
  FunctionalState ADC_ContinuousConvMode;
  uint32_t ADC_ExternalTrigConvEdge;
  uint32_t ADC_ExternalTrigConv;
  uint32_t ADC_DataAlign;
  uint32_t  ADC_ScanDirection;
}ADC_InitTypeDef;
```

参数 ADC_Resolution 是用来设置 ADC 的分辨率。前面讲解过，ADC 可以设置为 6 位，8 位，10 位或者是 12 位的分辨率，这里我们配置为 12 位采样，所以参数为 ADC_Resolution_12b。

参数 ADC_ContinuousConvMode 用来设置是否开启连续转换模式，因为是单次转换模式，所以我们选择不开启连续转换模式，DISABLE 即可。

参数 ADC_ExternalTrigConv 是用来设置启动规则转换组转换的外部事件，这里我们选择软件触发，选择值为 ADC_ExternalTrigConv_None 即可。

参数 DataAlign 用来设置 ADC 数据对齐方式是左对齐还是右对齐，这里我们选择右对齐方式 ADC_DataAlign_Right。

参数 ADC_ScanDirection 由软件设置和清除来选择通道序列中的通道扫描方向。这里我们只用到一个通道，所以向前或者向后扫描问题不大。

4）设置 ADC 的采样通道和采样时间。

ADC 的采样通道和采样时间可以通过下面这个函数实现：

```
void ADC_ChannelConfig(ADC_TypeDef* ADCx, uint32_t ADC_Channel, uint32_t ADC_SampleTime)
```

比如设置 ADC 的采集通道为通道 3，采样时间为 55.5 个 ADC 的采样时间周期，就可以这样设置：

```
ADC_ChannelConfig(ADC1, ADC_Channel_3  , ADC_SampleTime_55_5Cycles);
```

5）使能 ADC 并校准。

在设置完了以上信息后，我们就使能 AD 转换器，执行 AD 校准，注意这两步是必须的！不校准将导致结果很不准确。

使能指定的 ADC 的方法是：

```
ADC_Cmd(ADC1, ENABLE);
```

执行 ADC 校准的方法是：

```
ADC_GetCalibrationFactor(ADC1);
```

6）读取 ADC 值。

在上面的校准完成之后，ADC 就算准备好了。接下来我们要做的就是启动 ADC 转换。在转换结束后，读取 ADC 转换结果值就是了。

软件开启 ADC 转换的方法是：

```
ADC_StartOfConversion(ADC1);
```

开启转换之后，就可以获取转换 ADC 转换结果数据，方法是：

```
ADC_GetConversionValue(ADC1);
```

同时在 AD 转换中，我们还要根据状态寄存器的标志位来获取 AD 转换的各个状态信息。库函数获取 AD 转换的状态信息的函数是：

```
FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, uint32_t ADC_FLAG)
```

比如我们要判断 ADC 的转换是否结束，方法是：

```
while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));
```

通过以上几个步骤的设置，我们就能正常的使用 STM32 的 ADC 来执行

AD 转换操作了。

9.2硬件电路原理

本章实验是利用 AD 采集光强参数，并通过串口将采集的结果送到电脑的串口调试助手上进行查看，因此硬件连接部分由两部分构成，USART 和光敏电阻采集电路，其串口如何连接可以参考第 3 章内容，光敏电阻的连接原理图如图 6- 1 所示：

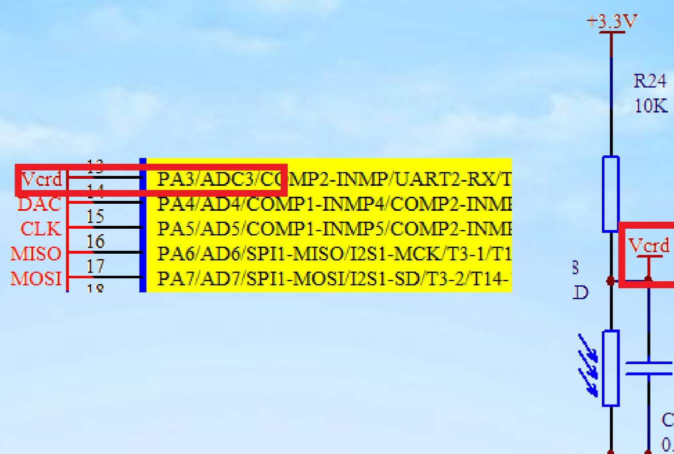


图 6- 1 连接原理图

从上可以看出，采集光强数据的是 ADC 的通道 3，而且在开发板上已经连接好了。

9.3软件程序与注解

双击打开我们资料包里的 ADC 的例程（路径为..\STM32+linux 资料包\例程\ADC 采集\user\ADC.Uv2），可以看到如图 6- 2 的工程：



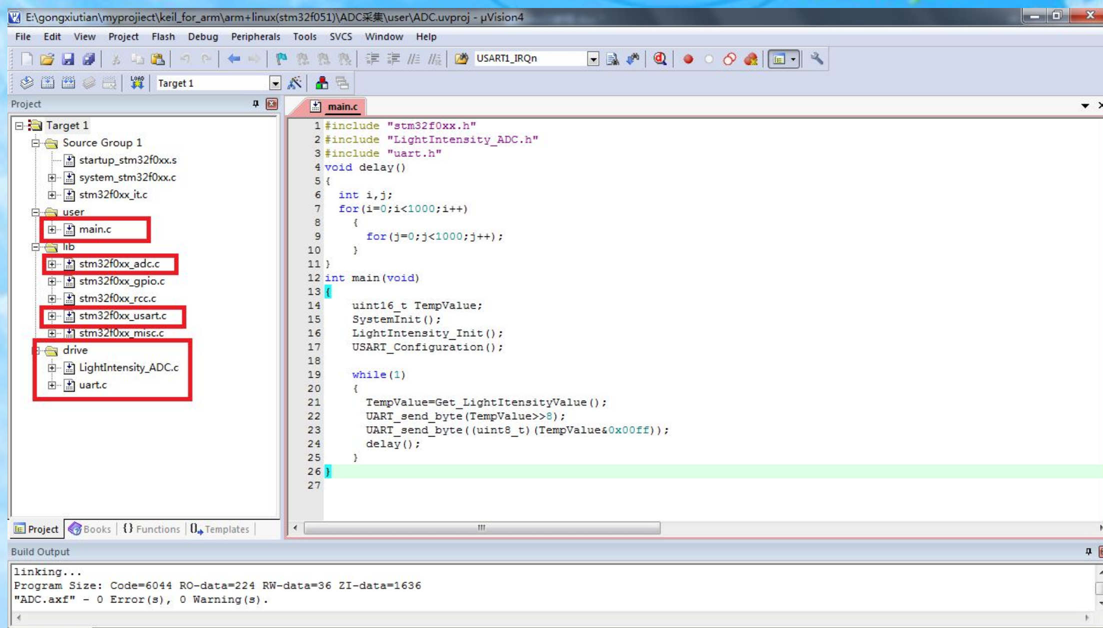


图 6- 2ADC 的例程

这里新编写的文件有 main.c 和 LightIntensity_ADC.c，因为用到了串口，所以把之前调试串口所编写的程序也添加了进来，如上图所示。主函数内容如下：

```
int main(void)
{
    uint16_t TempValue;
    SystemInit();
    LightIntensity_Init();
    USART_Configuration();

    while(1)
    {
      TempValue=Get_LightItensityValue();
      UART_send_byte(TempValue>>8);
      UART_send_byte((uint8_t)(TempValue&0x00ff));
      delay();
    }
}
```

从主函数的字面意思可以看出来该工程的功能，先是对系统进行初始化，实际上是对时钟的初始化，这里我们用的是内部时钟，所以这个函数实际上不写也是可以的。然后是对光敏电阻的采集进行初始化，实际上是对 ADC 的通道 3 进行初始化配置，也是我们要将的重点内容，其具体函数定义在 LightIntensity_ADC.c 里，内容为：



```c
void LightIntensity_Init(void)
{
  ADC_InitTypeDef     ADC_InitStruct;
  GPIO_InitTypeDef    GPIO_InitStruct;
    /* ADC1 DeInit */


  /* Enable  GPIOA clock */
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
  /* ADC1 Periph clock enable */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
  ADC_DeInit(ADC1);

  /* Configure PA.03  as analog input */
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_3;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOA, &GPIO_InitStruct);        // PC1,输入时不用设置速率

  /* Initialize ADC structure */
  ADC_StructInit(&ADC_InitStruct);

  /* Configure the ADC1 in continous mode withe a resolutuion equal to 12 bits  */
  ADC_InitStruct.ADC_Resolution = ADC_Resolution_12b;
  ADC_InitStruct.ADC_ContinuousConvMode = DISABLE;
  ADC_InitStruct.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
  ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStruct.ADC_ScanDirection = ADC_ScanDirection_Backward;
  ADC_Init(ADC1, &ADC_InitStruct);



  /* Convert the ADC1 Vref  with 55.5 Cycles as sampling time */
  ADC_ChannelConfig(ADC1, ADC_Channel_3  , ADC_SampleTime_55_5Cycles);
//    ADC_VrefintCmd(ENABLE);

  /* ADC Calibration */
  ADC_GetCalibrationFactor(ADC1);
  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);

  /* Wait the ADCEN falg */
  while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_ADEN));

  /* ADC1 regular Software Start Conv */
  ADC_StartOfConversion(ADC1);
}
```

首先是使能了 ADC3 所在的 GPIO 和 ADC 的时钟，然后将此引脚配置成模拟输入模式，然后配置 ADC 为 12 位采样精度，单次采样模式，没有外部触发，数据右对齐，向下扫描模式（这个是可以更改的），设置采集的通道为第三个通道，采样的时间为 55.5 个 ADC 周期。最后是进行校准和使能 ADC，并等待 ADC 启动完全。

回到主函数，当上面这个主函数执行完毕之后，又初始化了串口，然后在 while 循环里，我们就可以一直的采集光强数据，并将其通过串口发送出去：

```c
TempValue=Get_LightItensityValue();
UART_send_byte(TempValue>>8);
UART_send_byte((uint8_t)(TempValue&0x00ff));
delay();
```

其中 Get_LightItensityValue() 是获取光强数值的函数，其函数定义在 LightIntensity_ADC.c 里，内容如下：



```
uint16_t Get_LightItensityValue(void)
{
    ADC_StartOfConversion(ADC1);
    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));
    return ADC_GetConversionValue(ADC1);
}
```

调用开启转换函数(定义在库函数 stm32f0xx_adc.c 里)，然后等待转换结束，然后用 ADC_GetConversionValue(ADC1)获取转换结果，并返回，这样在主函数里，用一个读取到这个结果，然后将其上传至串口就可以了。

9.4程序下载与测试

本章实验是利用 AD 采集光强参数，并通过串口将采集的结果送到电脑的串口调试助手上进行查看。

插上串口，将程序下载到开发板（参考串口下载），打开串口调试助手，选择好 com 口之后，波特率设置为 115200，这里需要将串口调试助手的 Receive As Hex 勾选，因为我们上传的是 16 进制数据，而并非是 ASCII 码，点击 open，就可以看到接收窗口有数据产生了（如图 6- 3）：

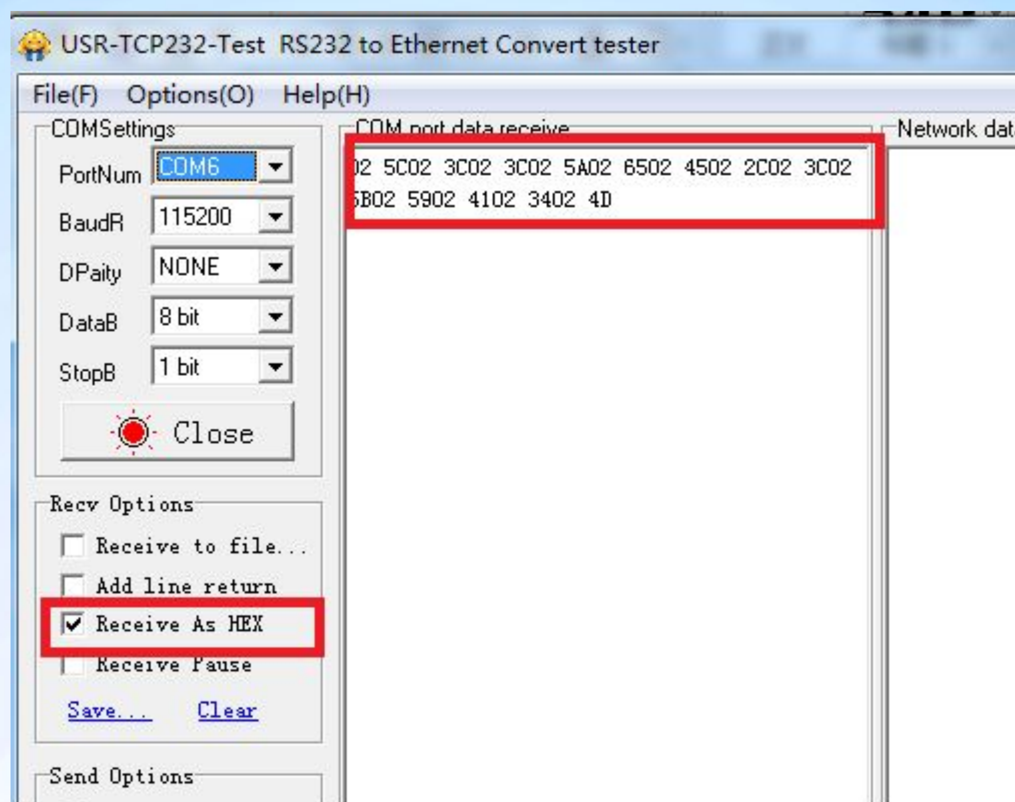


图 6- 3 串口显示

用手捂住串口，可以看到数值比原来的变大了如图 6- 4：



图 6- 4 数值变化

第 10 章 I2C 学习与 EEPROM 读写

10.1 I2C 原理简介

IIC(Inter－Integrated Circuit)总线是一种由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。它是由数据线 SDA 和时钟 SCL 构成的串行总线，可发送和接收数据。在 CPU 与被控 IC 之间、IC 与 IC 之间进行双向传送，高速 IIC 总线一般可达 400kbps 以上。I2C 总线在传送数据过程中共有三种类型信号， 它们分别是：开始信号、结束信号和应答信号。
开始信号：SCL 为高电平时，SDA 由高电平向低电平跳变，开始传送数据。
结束信号：SCL 为高电平时，SDA 由低电平向高电平跳变，结束传送数据。
应答信号：接收数据的 IC 在接收到 8bit 数据后，向发送数据的 IC 发出特定的低电平脉冲，表示已收到数据。CPU 向受控单元发出一个信号后，等待受控单元发出一个应答信号，CPU 接收到应答信号后，根据实际情况作出是否继续传递信号的判断。若未收到应答信号，由判断为受控单元出现故障。

这些信号中，起始信号是必需的，结束信号和应答信号，都可以不要。IIC 总线时序图如图 7- 1 所示：

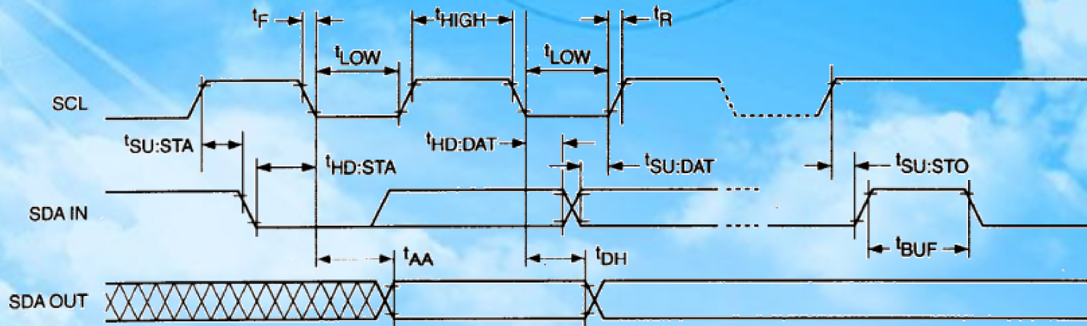


图 7- 1IIC 总线时序图

开发板板载的 EEPROM 芯片型号为 24C08。该芯片的总容量是 1024 个字节，该芯片通过 IIC 总线与外部连接，我们本章就通过 STM32 来实现 24C08 的读写。STM32 的 IIC 比较复杂，这里只能大概的讲一下，欲想知道更详细的内容，请参考库函数以及芯片的参考手册。

对 I2C 总线的操作需要了解的是三点：I2C 的初始化、对 I2C 器件的读操作和对 I2C 器件的写操作。

1）I2C 的初始化。首先要打开对 2C 相对应的 GPIO 口（I2C2 相对应的 IO 引脚是 PF5 和 PF6）和 I2C2 的时钟：

```
/* Enable  GPIOF clock */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOF, ENABLE);
/*!< sEE_I2C Periph clock enable */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C2 , ENABLE);
```

对 GPIO 进行初始化，要配置成复用的模式，配置 GPIO 的端口复用模式可以用这个函数来实现：

```
void GPIO_PinAFConfig(GPIO_TypeDef* GPIOx, uint16_t GPIO_PinSource, uint8_t GPIO_AF)
```

第一个参数是对应的 IO 口；第二个是对应的引脚；第三个是对应的模式，其值可以是以下几种：

```
@arg GPIO_AF_0:WKUP, EVENTOUT, TIM15, SPI1, TIM17,MCO, SWDAT, SWCLK, TIM14,
               BOOT,USART1, CEC, IR_OUT, SPI2
@arg GPIO_AF_1:USART2, CEC, Tim3, USART1, USART2,EVENTOUT, I2C1, I2C2, TIM15
@arg GPIO_AF_2:TIM2, TIM1, EVENTOUT, TIM16, TIM17.
@arg GPIO_AF_3:TS, I2C1, TIM15, EVENTOUT
@arg GPIO_AF_4:TIM14.
@arg GPIO_AF_5:TIM16, TIM17.
@arg GPIO_AF_6:EVENTOUT.
@arg GPIO_AF_7:COMP1 OUT, COMP2 OUT
```

因此我们要将 I2C2 所对应的引脚配置成复用模式，可以这样配置：

```
/* Connect PXx to I2C_SCL*/
 GPIO_PinAFConfig( GPIOF , GPIO_PinSource6, GPIO_AF_1);
 /* Connect PXx to I2C_SDA*/
  GPIO_PinAFConfig( GPIOF ,GPIO_PinSource7, GPIO_AF_1);
```

然后是对 I2C 的初始化，同样也是通过配置结构体来实现，这个结构体的原型如下图所示：

```c
typedef struct
{
  uint32_t I2C_Timing;              /*!< Specifies the I2C_TIMINGR_register value.
                                         This parameter must be set by referring to I2C_Timing_Config_Tool*/

  uint32_t I2C_AnalogFilter;        /*!< Enables or disables analog noise filter.
                                         This parameter can be a value of @ref I2C_Analog_Filter*/

  uint32_t I2C_DigitalFilter;       /*!< Configures the digital noise filter.
                                         This parameter can be a number between 0x00 and 0x0F*/

  uint32_t I2C_Mode;                /*!< Specifies the I2C mode.
                                         This parameter can be a value of @ref I2C_mode*/

  uint32_t I2C_OwnAddress1;         /*!< Specifies the device own address 1.
                                         This parameter can be a 7-bit or 10-bit address*/

  uint32_t I2C_Ack;                 /*!< Enables or disables the acknowledgement.
                                         This parameter can be a value of @ref I2C_acknowledgement*/

  uint32_t I2C_AcknowledgedAddress; /*!< Specifies if 7-bit or 10-bit address is acknowledged.
                                         This parameter can be a value of @ref I2C_acknowledged_address*/
}I2C_InitTypeDef;
```

这个结构体包括 I2C 的时序，模拟滤波，数字滤波，发送的模式，7 位或是 10 位的地址选择，是否相应应答，等信息。

最后使能 I2C：

```c
/* I2C Peripheral Enable */
I2C_Cmd(I2C2, ENABLE);
```

2）对 I2C 进行读或写操作。对 I2C 进行读或写操作之前，需要对这个操作进行配置，其配置的函数为：

```c
void I2C_TransferHandling(I2C_TypeDef* I2Cx, uint16_t Address,
uint8_t Number_Bytes, uint32_t ReloadEndMode, uint32_t StartStopMode)
```

在每次读或者写之前都要调用这个函数进行配置，其参数有 5 个：

- I2Cx 可以是 I2C1 或 I2C2
- Address 指定要编程的从地址
- Number_Bytes 是要写入从器件的字节数
- ReloadEndMode 在 I2C 起始条件生成的新状态，它的值可以是：I2C_Reload_Mode(重装载模式，即传输 Number_Bytes 个字节后，传输并不结束)、I2C_AutoEnd_Mode(自动结束模式：Number_Bytes 个数据传输完后，会自动发送一个停止条件)、I2C_SoftEnd_Mode(软件结束模式：当 Number_Bytes 个数据传输完毕后，需要编程检测标志位来确定是否结束操作，同时需要软件决定是否设置停止位)。
- StartStopMode 配置起始或者是停止条件，它的值可以是：I2C_No_StartStop(不产生起始条件和停止位)、I2C_Generate_Stop(产生停止位)、I2C_Generate_Start_Read(主机产生一个读的请求)、I2C_Generate_Start_Write(主机产生一个写的请求)。

配置好上一个函数之后就可以对从器件进行读或写的操作了，其函数如下所示：

```c
void I2C_SendData(I2C_TypeDef* I2Cx, uint8_t Data)
uint8_t I2C_ReceiveData(I2C_TypeDef* I2Cx)
```

最后要根据状态寄存器判断读或写的动作是否完成，通常用 I2C_GetFlagStatus()函数来完成。

```c
FlagStatus I2C_GetFlagStatus(I2C_TypeDef* I2Cx, uint32_t I2C_FLAG)
```

I2C_FLAG 的可以是如下的值：

*@ARGI2C_FLAG_TXE：发送数据寄存器空
*@ARGI2C_FLAG_TXIS：发送中断状态
*@ARGI2C_FLAG_RXNE：接收数据寄存器不为空
*@ARGI2C_FLAG_ADDR：地址匹配（从模式）
*@ARGI2C_FLAG_NACKF：收到 NACK 标志
*@ARGI2C_FLAG_STOPF：停止检测标志
*@ARGI2C_FLAG_TC：传输完成（主模式）
*@ARGI2C_FLAG_TCR：传输完成重装
*@ARGI2C_FLAG_BERR：总线错误
*@ARGI2C_FLAG_ARLO：仲裁丢失
*@ARGI2C_FLAG_OVR：溢出/欠载
*@ARGI2C_FLAG_PECERR：PEC 在接收错误
*@ARGI2C_FLAG_TIMEOUT：超时或 TLOW 检测标志
*@ARGI2C_FLAG_ALERT：SMBus 报警
*@ARGI2C_FLAG_BUSY：总线忙

了解了以上函数，我们就可以进行简单的 I2C 的读写操作了。下图是 i2c 主机从从机读数据的过程：



首先是开始信号，然后发送从机地址，写选通，给一个应答信号。然后写入要访问的地址。再次应答。从新开始信号，发送从机地址，从机地址是外挂总线地址。然后读选通，主机等待从机的应答信号，当接收应答后，开始读取数据。

10.2 硬件电路原理

EEPROM 与 STM32 单片机的硬件连接图如下图 7- 2 所示：

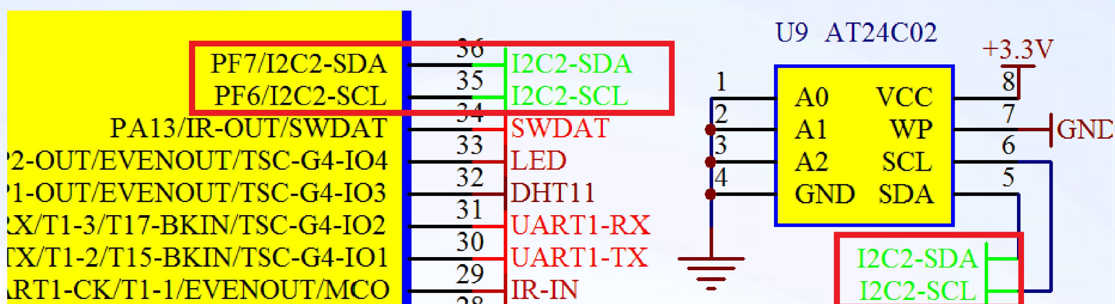


图 7- 2 硬件连接图

在开发板上这个是连接好的，所以可以直接操作。



10.3 软件程序与注解

双击打开我们资料包里液晶屏的例程(路径为..\STM32+linux 资料包\例程\i2c 操作 24c02(库函数）\user\i2c_eeprom.Uv2)，可以看到如图 7- 3 的工程：

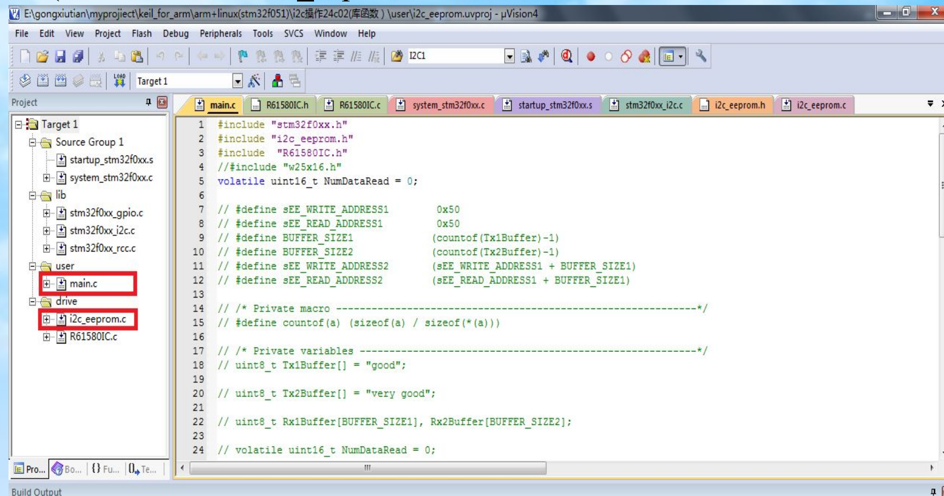


图 7- 3 液晶屏的例程

工程需要包括的库文件有 stm32f0xx_gpio.c 、 stm32f0xx_rcc.c 和 stm32f0xx_i2c.c，需要用户编辑的文件有主函数 main.c 和驱动文件 i2c_eeprom.c，因为用到了液晶屏来显示字符，所以把之前的液晶屏驱动文件 R61580IC.c 也添加了进来，

我们先从主函数开始分析，主函数包括的头文件有：

```
#include "stm32f0xx.h"
#include "i2c_eeprom.h"
#include  "R61580IC.h"
```

接下来定义了两个数组，用来存放写进或者读出 24c08 里的数据,其中 temp1 里存放的是写入 24c08 里的字符，temp2 里存放要接收的字符。

```
char temp1[]="microcloud";
char temp2[11];
```

然后是主函数内容：

```
int main(void)
{
    SystemInit();
    R61580_init();   // 液晶显示器初始化
    LCD_Clear(ORANGE);   // 全屏显示白色
    POINT_COLOR =BLACK; // 定义笔的颜色为黑色
   BACK_COLOR = WHITE  ;   // 定义笔的背景色为白色
    I2C_EE_Init();

    LCD_ShowString(2,2,"Experimental:");
    LCD_ShowString(20,20,"i2c read 24c02 Experiment");
    sEE_WriteBuffer(temp1, 0x050, 11);
    LCD_ShowString(2,40,"24c02 write value:");
    LCD_ShowString(50,60,(char *)temp1);
    NumDataRead=11;
    sEE_ReadBuffer(temp2, 0x050,(uint16_t *)(&NumDataRead));
    LCD_ShowString(2,80,"24c02 read value:");
    LCD_ShowString(50,100,(char *)temp2);

      if(*temp1 != *temp2)
      {
         LCD_ShowString(50,120,"read EEROR");
      }
    else
      {
          LCD_ShowString(50,120,"read success");
      }

    while(1)
     {}
  }
```

从函数的名字我们可以大约的分辨出函数的功能，首先是系统的初始化SystemInit，因为我们没有设置外部时钟，所以在这个函数里我们要 SetSysClock()的 else 语句里将外部时钟关掉，修改如下图所示：

```
  else
  { /* If HSE fails to start-up, the application will have wrong clock
         configuration. User can add here some code to deal with this error */
    RCC->CR &= ~((uint32_t)RCC_CR_HSEON);
  }
```

然后是液晶屏的初始化 R61580_init()，这个不细说，详细的情况请参考第 4 章内容。LCD_Clear(ORANGE);是将整个 LCD 界面刷成橙色，然后是定义画笔颜色和背景：

```
POINT_COLOR =BLACK; // 定义笔的颜色为黑色
BACK_COLOR = WHITE  ;  // 定义笔的背景色为白色
```

LCD_ShowString(2,2,"Experimental:");等是在液晶屏的相应位置显示字符串，这里我们主要需要分析的函数有三个：

I2C_EE_Init();

sEE_WriteBuffer(temp1, 0x050, 11);

sEE_ReadBuffer(temp2, 0x050,(uint16_t *)(&NumDataRead));

1）I2C_EE_Init();是对 I2C 的初始化。主要是使能相应的时钟，配置 GPIO 的复用模式，配置 I2C 的参数（结构体）和使能 I2C 等，前面有详细介绍，这里不多做介绍了。

2)void sEE_WriteBuffer(uint8_t* pBuffer, uint16_t WriteAddr, uint16_t NumByteToWrite)是向 24c08 器件里写数据。

pBuffer 是要写进去的数据。数据指向一个字节数组。

WriteAddr 是器件的内部地址。并非是器件地址，这里我们说一下 24C08 的器件地址和字地址的区别。

器件寻址

起始条件使能芯片读写操作后，EEPROM都要求有8位的器件地址信息（见图8）。

器件地址信息由"1"、"0"序列组成，前4位如图中所示，对于所有串行EEPROM都是一样的。

对于24C02/32/64，随后3位A2、A1和A0为器件地址位，必须与硬件输入引脚保持一致。

对于24C04，随后2位A2和A1为器件地址位，另1位为页地址位。A2和A1必须与硬件输入引脚保持一致，而A0是空脚。

对于24C08，随后1位A2为器件地址位，另2位为页地址位。A2必须与硬件输入引脚保持一致，而A1和A0是空脚。

对于24C16，无器件地址位，3位都为页地址位，而A2、A1和A0是空脚。

器件地址信息的LSB为读/写操作选择位，高为读操作，低为写操作。

若比较器件地址一致，EEPROM将输出应答"0"。如果不一致，则返回到待机状态。



从 24C08 的芯片资料上可以了解到，器件地址主要是用于总线上多于一个器件的时候区分从器件的地址，一个总线最多可以连接两个 24c08 器件，主要由 A2 位来区分，而对芯片的页写操作如下图 7- 4 所示：

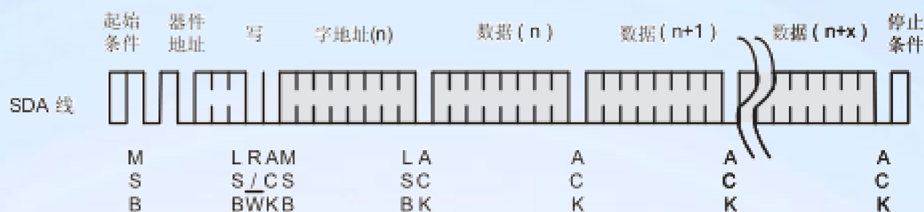


图 7- 4 页写操作

在起始位之后，先写的是器件地址，在软件上我们通过函数

```
void I2C_TransferHandling(I2C_TypeDef* I2Cx, uint16_t Address,
uint8_t Number_Bytes, uint32_t ReloadEndMode, uint32_t StartStopMode)
```

来将器件地址送上总线，然后再发送一个字节

```
/* Send memory address */
I2C_SendData(sEE_I2C, (uint8_t)WriteAddr);
```

就是从器件的字地址了，也就是 24C08 的内部寻址，24C08 是 8K 的容量，能够存储 1024 个字节，所以内部寻址空间是 1K，显然 7 位的字地址不够，因此器件地址的低 2 位属于页寻址部分，如。。。图所示。这里很显然，我们的例程的地址范围是 0~255，只能进行 256 个字节以内的读写。对于更多字节的读写，请参考芯片资料和参考手册进行学习。

NumByteToWrite 是要写进去的字节数。EEPROM 是按页操作的，一次性最多写一页，24C08 定义一页是 16 个字节，所以向器件读或者写都要按页操作。



3)uint32_t sEE_ReadBuffer(uint8_t* pBuffer, uint16_t ReadAddr, uint16_t* NumByteToRead) 是向器件里读数据， 其三个参数(pBuffer, ReadAddr, NumByteToRead)和写操作的参数是一样的，这里不多作介绍了。

10.4 程序下载与测试

本工程实现的功能是：STM32 读写 EEPROM。

参照串口下载将程序下载到开发板，就可以如下图 7- 5 界面：

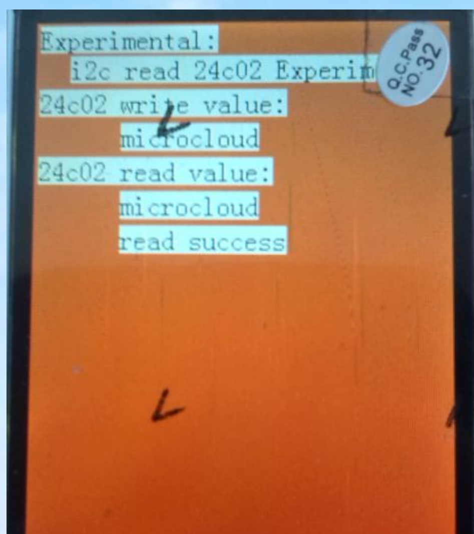


图 7- 5 24C08 读写成功

这说明对 24C08 的读写成功了。

第 11 章 SPI 读写串行 FLASH

11.1 SPI 原理简介

SPI，串行外围设备接口。主要应用在 EEPROM，FLASH，实时时钟，AD 转换器，还有数字信号处理器和数字信号解码器之间，是一种高速的，全双工，同步的串行通信总线。

通常 SPI 通过 4 个引脚与外部器件相连：

● MISO：（主入从出）主设备数据输入，从设备数据输出。该引脚在从模式下发送数据，在主模式下接收数据。

● MOSI：（主出从入）主设备数据输出，从设备数据输入。该引脚在主模式下发送数据，在从模式下接收数据。

● SCK：串口时钟。

●NSS：从设备片选信号，由主设备控制。只有在片选选通时，才可以对从机进行读或写操作，一般情况下低电平有效。

一般情况下，SPI 也可以仅有 3 条线组成，即 SCK 时钟线，SDA 数据线和 CS 片选，SDA 可配置为双向数据传输。

SPI 总线的通信时序：



从机在时钟线的上升沿或者下降沿锁存数据。

SPI 主要特点有：可以同时发出和接收串行数据；可以当作主机或从机工作；提供频率可编程时钟；发送结束中断标志；写冲突保护；总线竞争保护等。

SPI 功能框图及工作原理如图 8- 1 所示：

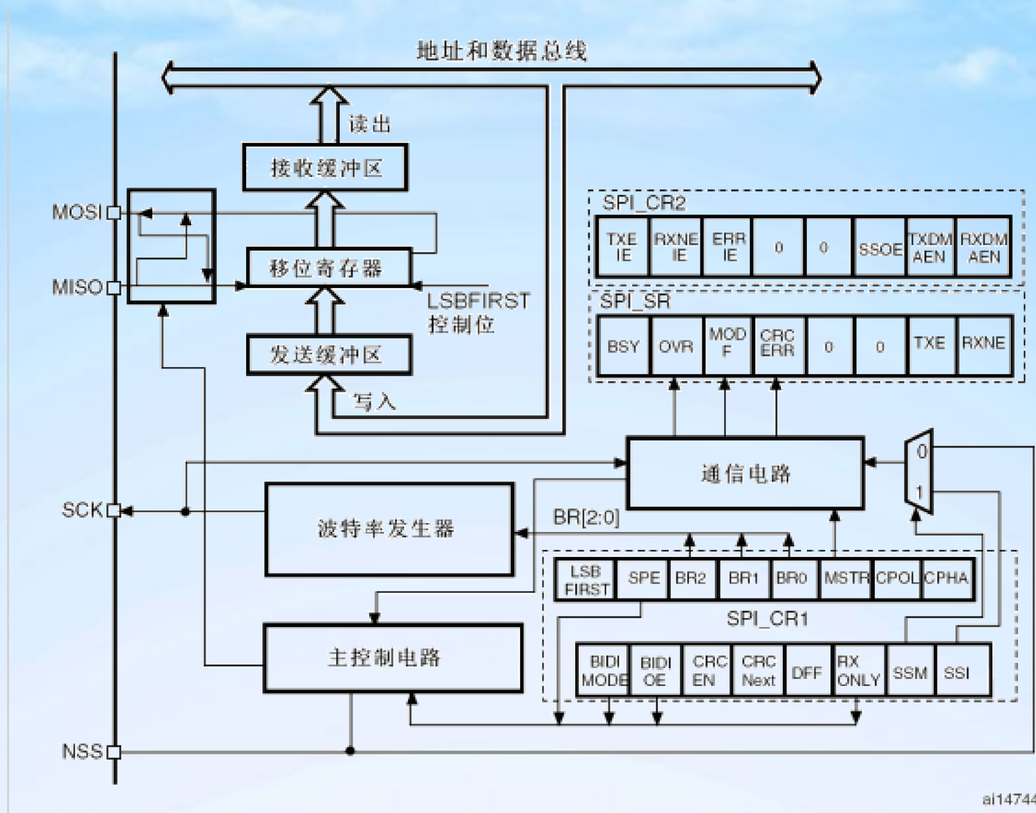


图 8- 1SPI 功能框图

全双工模式下：当写入数据到 SPI_DR 寄存器(发送缓冲器)后，传输开始；

在传送第一位数据的同时，数据被并行地从发送缓冲器传送到 8 位的移位寄存器中，然后按顺序被串行地移位送到 MOSI 引脚上；

与此同时，在 MISO 引脚上接收到的数据，按顺序被串行地移位进入 8 位的移位寄存器中，然后被并行地传送到 SPI_DR 寄存器(接收缓冲器)中。

11.2 FLASH 原理简介

串行 FLASH 采样 25Q32。采用的 Flash 存储芯片的存储容量是 4M 字节。

Flash 存储芯片可以为用户提供存储解决方案，适合代码下载应用，例如存

储声音、文本和数据。工作电压在 2.7V 至 3.6V 之间。W25Q32FVSIG Flash 存储芯片有 16384 可编程页，每页 256 个字节，还具有 1024 个可擦除“扇区”或 64 个可擦除“块”。“页编程指令”每次编程 256 个字节，“扇区擦除指令”每次擦除 16 页，“块擦除指令”每次擦除 256 页，“整片擦除指令”每次擦除整个芯片。

1）根据图 8- 2 中的 Flash 存储芯片封装，表 8- 1 介绍了 Flash 存储芯片各个引脚的功能。

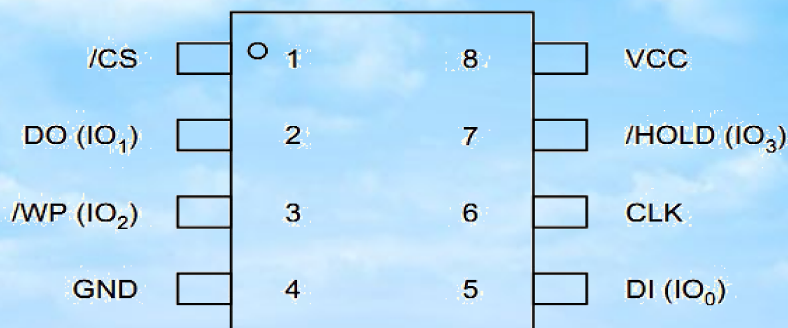


图 8- 2Flash 存储芯片封装

表 8- 1Flash 存储芯片引脚介绍

引脚号	引脚名称	输入输出（I/O）	功能
1	/CS	I	芯片选择
2	DO	I/O	数据输出
3	/WP	I/O	写保护
4	GND		地
5	DI	I/O	数据输入
6	CLK	I	时钟
7	/HOLD	I/O	保护
8	VCC		电源

2）由于在私有云系统中对 Flash 存储芯片的操作是已经将图片数据烧入 Flash 存储器中，所以针对 Flash 存储器的操作主要是“读数据指令”操作。

“读数据指令”允许一个字节或一个以上字节被读出。拉低/CS 引脚，把 03h 通过 DI 引脚送到芯片，然后送入 24 位地址，以上数据在 CLK 上升沿被芯片采集。芯片接收完 24 位地址后，就会把相应地址的数据在 CLK 引脚下降沿从 DO 引脚送出，且高位在前。当读完这个地址的数据后，地址自动增加，然后通过引脚把下一个地址的数据送出，形成数据流。

当数据读完后，把/CS 引脚拉高，“读数据指令”结束。当芯片在执行“页编程指令”、“擦除指令”和“读状态寄存器指令”的周期内，“读数据指令”不起作用。

11.3 硬件电路原理

图 8- 3 为私有云系统中的 Flash 存储芯片硬件图

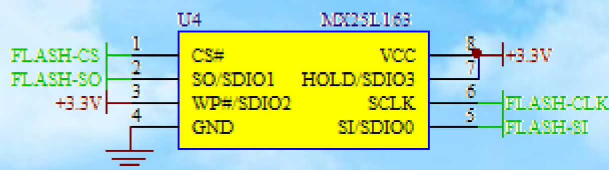


图 8- 3Flash 存储芯片硬件原理图

由于写保护引脚 WP 可以被用来保护状态寄存器不被意外改写，写保护引脚接 3.3V。保持引脚 HOLD 拉高器件正常工作，所以 HOLD 引脚接 3.3V。

FLASH 与 STM32 单片机的硬件连接图如下图 8- 4 所示：

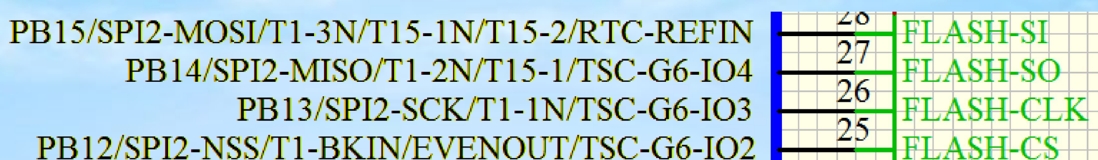


图 8- 4STM32 管脚图

在开发板上这个是连接好的，所以可以直接操作。

11.4 软件程序与注解

双击打开我们资料包里 FLASH 的例程（路径为..\STM32+linux 资料包\例程\i2c 操作 24c02(库函数）\user\i2c_eeprom.Uv2），可以看到如下图 8- 5 工程：

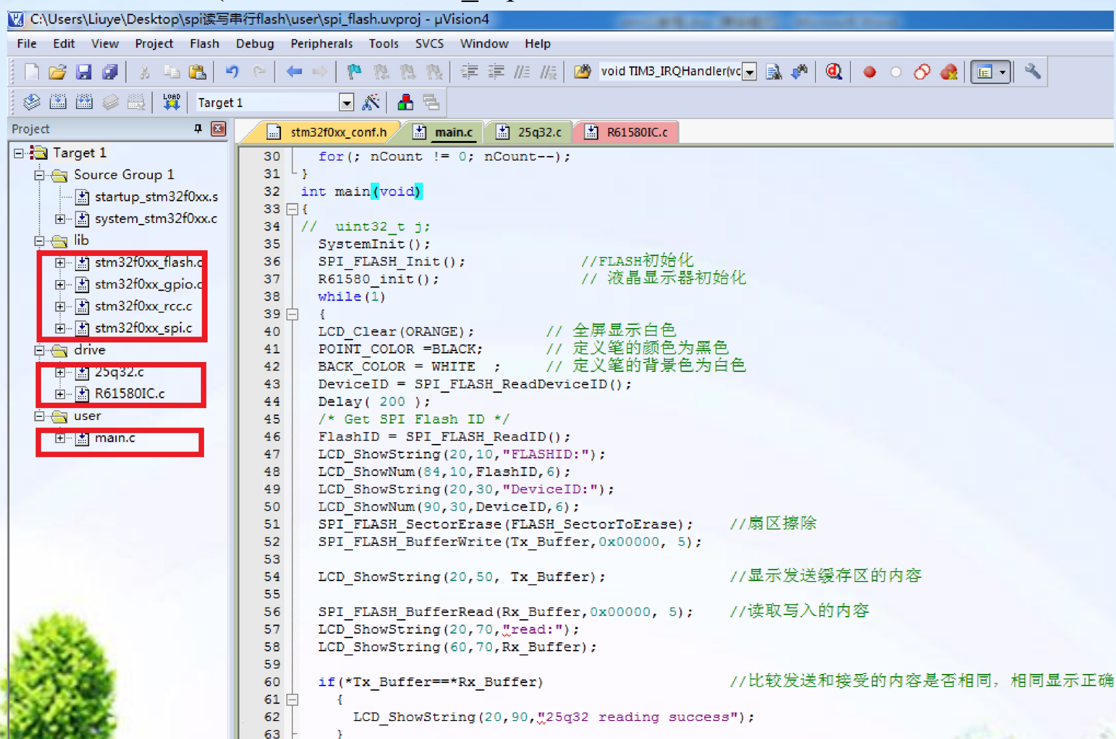


图 8- 5FLASH 工程

主函数实现的功能：液晶屏显示从 FLASH 中读取的 FLASHID 和 DeviceID；



扇区擦除完毕，判断向 flash 中写的数据和读取的数据相同。

首先是对 SPI_FLASH 进行初始化，包括初始化几个方面：

(1) GPIO 和 SPI 的结构体及时钟配置

```
GPIO_InitTypeDef  GPIO_InitStruct;
SPI_InitTypeDef   SPI_InitStruct;
RCC_AHBPeriphClockCmd( RCC_AHBPeriph_GPIOB, ENABLE);//设置GPIO时钟
RCC_APB1PeriphClockCmd(FLASH_SPI2, ENABLE);         //配置SPI时钟
```

(2) SPI1 在没有重映射的条件下各引脚初始化：

```
// 25Q32-CLK ,25Q32-DO,25Q32-DIO
GPIO_InitStruct.GPIO_Pin = FLASH_SCK_PIN;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;          //IO复用
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_Level_3;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;        //推挽
GPIO_InitStruct.GPIO_PuPd  = GPIO_PuPd_UP;         //上拉
GPIO_Init(FLASH_SCK_PORT, &GPIO_InitStruct);

GPIO_InitStruct.GPIO_Pin = FLASH_MISO_PIN;
GPIO_Init(FLASH_MISO_PORT, &GPIO_InitStruct);
GPIO_InitStruct.GPIO_Pin =FLASH_MOSI_PIN;
GPIO_Init(FLASH_MOSI_PORT, &GPIO_InitStruct);
GPIO_PinAFConfig(FLASH_SCK_PORT, FLASH_SCK_SOURCE, FLASH_SCK_AF);//某端口设置为复用
GPIO_PinAFConfig(FLASH_MISO_PORT, FLASH_MISO_SOURCE, FLASH_MISO_AF);
GPIO_PinAFConfig(FLASH_MOSI_PORT, FLASH_MOSI_SOURCE, FLASH_MOSI_AF);

GPIO_InitStruct.GPIO_Pin =FLASH_CS_PIN;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_Level_3;
GPIO_Init(FLASH_CS_PORT, &GPIO_InitStruct);
```

★ 在配置 GPIO 的 25Q32-CLK ,25Q32-DO,25Q32-DIO 时将其配置为复用输出,在复用功能下面,输入输出的方向,完全由内部控制.不需要程序处理。

配置片选信号线，并设为高电平。

(3) 配置 SPI2 的参数 SPI 的方向、工作模式、数据帧格式、CPOL、CPHA、NSS 软件还是硬件、SPI 时钟、数据的传输位、以及 CRC

```
SPI_InitStruct.SPI_Direction = SPI_Direction_2Lines_FullDuplex;//配置SPI方向
SPI_InitStruct.SPI_Mode = SPI_Mode_Master;                    //配置SPI工作模式
SPI_InitStruct.SPI_DataSize = SPI_DataSize_8b;                //配置数据帧格式
SPI_InitStruct.SPI_CPOL = SPI_CPOL_High;                      //配置时钟高电平稳态
SPI_InitStruct.SPI_CPHA = SPI_CPHA_2Edge;                     //配置时钟bit位捕获方式
SPI_InitStruct.SPI_NSS = SPI_NSS_Soft;                        //配置NSS管脚软件管理
SPI_InitStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_16;//设置波特率分频值，时钟
SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB;                //指定数据传输从高位开始
SPI_InitStruct.SPI_CRCPolynomial = 7;                          //指定用于CRC校验的值
SPI_Init(SPI2, &SPI_InitStruct);
SPI_RxFIFOThresholdConfig(SPI2, SPI_RxFIFOThreshold_QF);
SPI_Cmd(SPI2, ENABLE);
```

（1） 在 SPI 通信中，在全双工模式下，发送和接收是同时进行的。

（2） 数据帧宽度可编程：支持 8 位和 16 位数据传输格式。

（3） 时钟配置：数据传输的时钟来自主控制器的时钟脉冲，最常用的时钟设置是基于时钟极性 CPOL 和时钟相位 CPHA 两个参数。

a) CPOL=0，表示时钟的空闲状态为低电平

b) CPOL=1，表示时钟的空闲状态为高电平

c) CPHA=0，表示同步始终的第一个边沿（上升或者下降）数据被采样

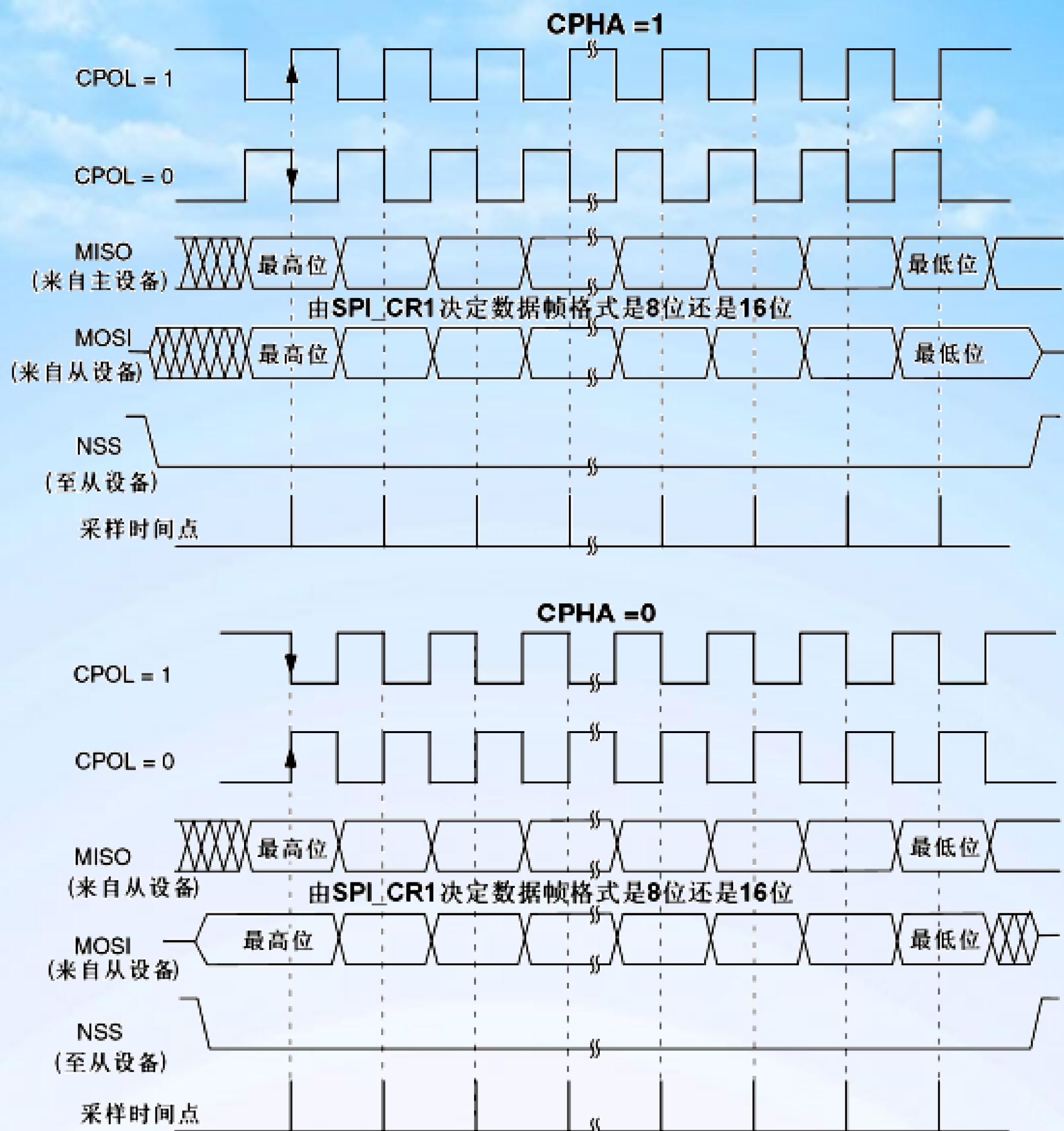
d) CPHA=1，表示同步始终的第二个边沿（上升或者下降）数据被采样

即 CPOL 和 CPHA 的设置决定了数据采样的时钟沿。

（4） SPI 四种工作模式，具体如图 8- 6：

SPI主机配置位		
CPOL	CPHA	模式
0	0	0
0	1	1
1	0	2
1	1	3

图 8- 6 四种工作模式



（5） 可作为主设备也可作从设备

（6） SPI 通信是串行发送或接收数据的，即一位一位的发送和接收、且传输一般是高位 MSB 在前，低位在后 LSB。

（7） 可编程的位传输速率：在主/从模式下，最大都可达 18MHZ。注意：在与被控制器件连接时，只用 SCK、MOSI、MISO 三条线即可，用软件控制 NSS 使之处于主机模式，通过 GPIO 选择被控器件即可

（8） 带中断性能的主模式故障或溢出错误标志位和 CRC 错误标志：

a) 主模式故障：在硬件模式下：主设备的 NSS 脚被拉低；在软件模式下，SSI 位被复位(SPI_SR 中的 MODF 被硬件置位)。

b) 溢出错误:从设备在接收新数据时，而前一个数据还没有被读出，即发出溢出错误。

（9） 拥有可靠通信的硬件 CRC 特性：在发送模式下 CRC 的值作为最后一个字节发送；对接收到的最后一个字节自动进行 CRC 校验（只使用与全双工通信，CRC8 算法用于 8 位帧格式，CRC16 算法用于 16 位帧格式）

接下来我们一起来分析一下 SPI_FLASH_SendByte 函数:由于主机 SPI 通信时，在发送和接受时是同时进行的,即发送完了一个字节的数据后，也应当接受到一个字节的数据

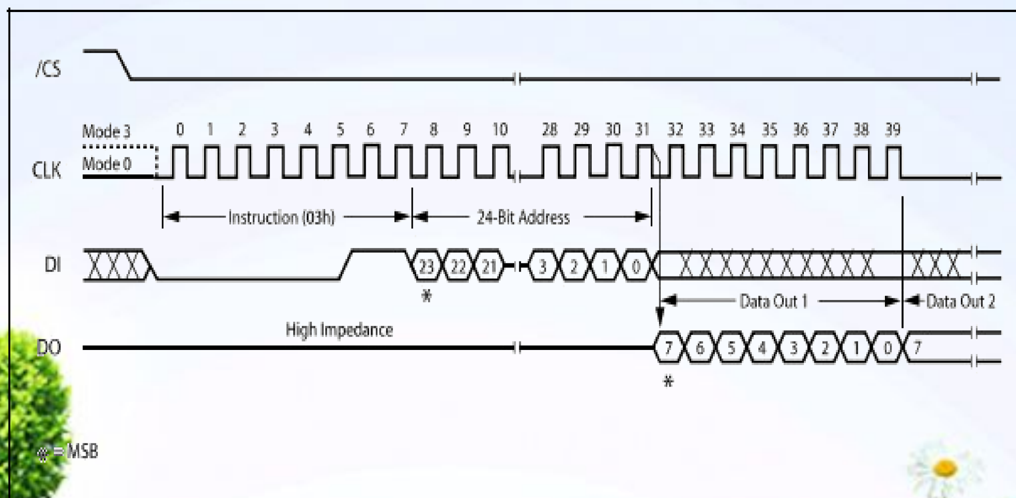
```
uint8_t SPI_FLASH_SendByte(uint8_t byte)
{
  while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE) == RESET);//判断是否发送完成
  SPI_SendData8(SPI2, byte);                                      //spi发送字节
  while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_RXNE) == RESET);//是否已经读取
  return SPI_ReceiveData8(SPI2);                                  //spi接收
}
```

注意：STM32F051 中的 SPI_DR 寄存器为 16 位的，SPIx->DR = (uint16_t)Data; 是发送 16 位数据。我们这里发送 8 位数据使用的是库函数。

```
void SPI_SendData8(SPI_TypeDef* SPIx, uint8_t Data)
{
  uint32_t spixbase = 0x00;
  /* Check the parameters */
  assert_param(IS_SPI_ALL_PERIPH(SPIx));
  spixbase = (uint32_t)SPIx;
  spixbase += 0x0C;
  *(__IO uint8_t *) spixbase = Data;
}
```

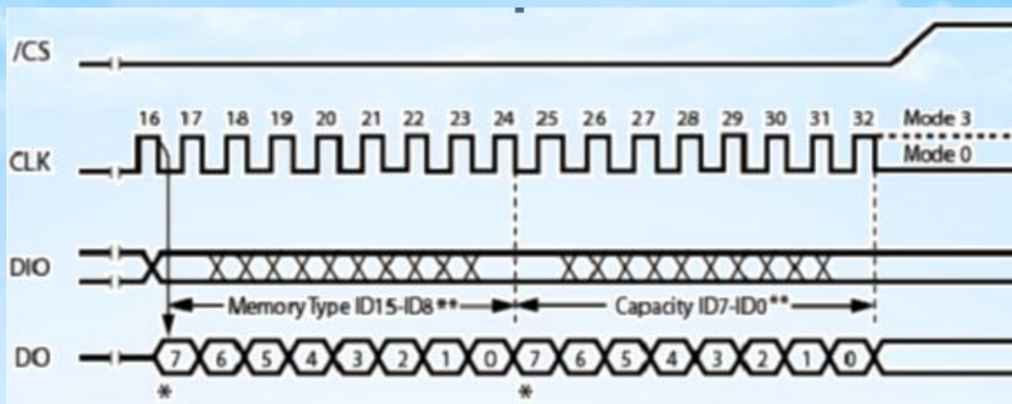
用 STM32F103 时用 SPI 发送 8 位数据就是直接 SPI1->DR = (uint8_t)Data;F103 的 SPI 会根据数据位的设置自动从 DR 寄存器的 LSB 截取数据。

有了以上的 STM32 SPI 驱动就可以去驱动 SPI 的任意器件，但是要注意 SPI 器件的操作驱动函数还要根据所用的 SPI 器件手册进行书写。在这里我们读和写 25Q32。下面写读取器件 ID 代码，参考 25Q32 代码参考手册中的时序图：



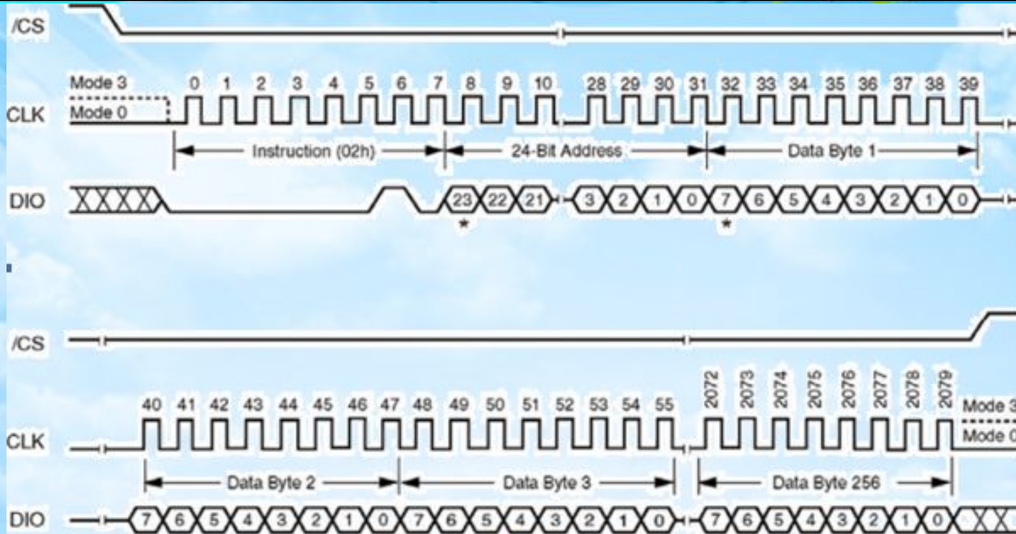
```
uint32_t SPI_FLASH_ReadDeviceID(void)
{
  uint32_t Temp = 0;
  SPI_FLASH_CS_LOW();          //使能
  SPI_FLASH_SendByte(W25X_DeviceID);
  SPI_FLASH_SendByte(Dummy_Byte);
  SPI_FLASH_SendByte(Dummy_Byte);
  SPI_FLASH_SendByte(Dummy_Byte);
  /* Read a byte from the FLASH */
  Temp = SPI_FLASH_SendByte(Dummy_Byte);
  /* Deselect the FLASH: Chip Select high */
  SPI_FLASH_CS_HIGH();
  return Temp;
}
```

读取制造 ID 参考时序图：




```
uint32_t SPI_FLASH_ReadID(void)
{
  uint32_t Temp = 0, Temp0 = 0, Temp1 = 0, Temp2 = 0;
  /* Select the FLASH: Chip Select low */
  SPI_FLASH_CS_LOW();
  /* Send "RDID " instruction */
  SPI_FLASH_SendByte(W25X_JedecDeviceID);//识别器件ID号
  /* Read a byte from the FLASH */
  Temp0 = SPI_FLASH_SendByte(Dummy_Byte);
  /* Read a byte from the FLASH */
  Temp1 = SPI_FLASH_SendByte(Dummy_Byte);
  /* Read a byte from the FLASH */
  Temp2 = SPI_FLASH_SendByte(Dummy_Byte);
  /* Deselect the FLASH: Chip Select high */
  SPI_FLASH_CS_HIGH();
  Temp = (Temp0 << 16) | (Temp1 << 8) | Temp2;
  return Temp;
}
```

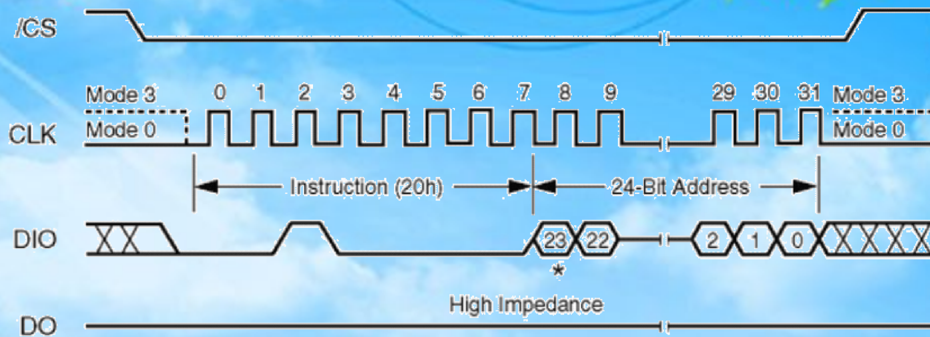
25Q32 页写参考时序图：




```
void SPI_FLASH_PageWrite(uint8_t* pBuffer, uint32_t WriteAddr, uint16_t NumByteToWrite)
{
  /* Enable the write access to the FLASH */
  SPI_FLASH_WriteEnable();
  /* Select the FLASH: Chip Select low */
  SPI_FLASH_CS_LOW();
  /* Send "Write to Memory " instruction */
  SPI_FLASH_SendByte(W25X_PageProgram);
  /* Send WriteAddr high nibble address byte to write to */
  SPI_FLASH_SendByte((WriteAddr & 0xFF0000) >> 16);
  /* Send WriteAddr medium nibble address byte to write to */
  SPI_FLASH_SendByte((WriteAddr & 0xFF00) >> 8);
  /* Send WriteAddr low nibble address byte to write to */
  SPI_FLASH_SendByte(WriteAddr & 0xFF);

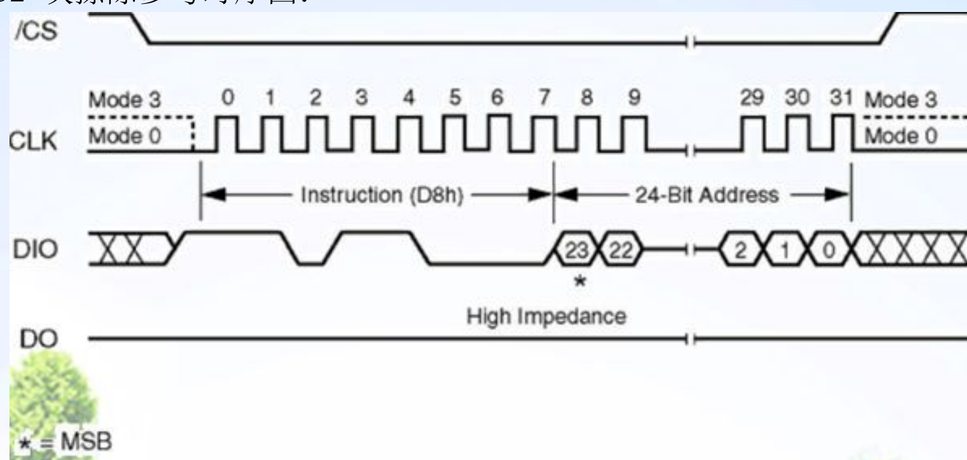
  if(NumByteToWrite > SPI_FLASH_PerWritePageSize)
  {
     NumByteToWrite = SPI_FLASH_PerWritePageSize;
     //printf("\n\r Err: SPI_FLASH_PageWrite too large!");
  }
  /* while there is data to be written on the FLASH */
  while (NumByteToWrite--)
  {
    /* Send the current byte */
    SPI_FLASH_SendByte(*pBuffer);
    /* Point on the next byte to be written */
    pBuffer++;
  }
  /* Deselect the FLASH: Chip Select high */
  SPI_FLASH_CS_HIGH();
  /* Wait the end of Flash writing */
  SPI_FLASH_WaitForWriteEnd();
}
```

25Q32 扇区擦除时序图：




```
void SPI_FLASH_SectorErase(uint32_t SectorAddr)
{
  /* Send write enable instruction */
  SPI_FLASH_WriteEnable();
  SPI_FLASH_WaitForWriteEnd();
  /* Sector Erase */
  /* Select the FLASH: Chip Select low */
  SPI_FLASH_CS_LOW();
  /* Send Sector Erase instruction */
  SPI_FLASH_SendByte(W25X_SectorErase);
  /* Send SectorAddr high nibble address byte */
  SPI_FLASH_SendByte((SectorAddr & 0xFF0000) >> 16);
  /* Send SectorAddr medium nibble address byte */
  SPI_FLASH_SendByte((SectorAddr & 0xFF00) >> 8);
  /* Send SectorAddr low nibble address byte */
  SPI_FLASH_SendByte(SectorAddr & 0xFF);
  /* Deselect the FLASH: Chip Select high */
  SPI_FLASH_CS_HIGH();
  /* Wait the end of Flash writing */
  SPI_FLASH_WaitForWriteEnd();
}
```

25Q32 块擦除参考时序图：





```
void SPI_FLASH_BulkErase(void)
{
  /* Send write enable instruction */
  SPI_FLASH_WriteEnable();
  /* Bulk Erase */
  /* Select the FLASH: Chip Select low */
  SPI_FLASH_CS_LOW();
  /* Send Bulk Erase instruction  */
  SPI_FLASH_SendByte(W25X_ChipErase);
  /* Deselect the FLASH: Chip Select high */
  SPI_FLASH_CS_HIGH();
  /* Wait the end of Flash writing */
  SPI_FLASH_WaitForWriteEnd();
}
```

至此 FLASH 的基本操作就已经完成。

11.5 程序下载与测试

本工程实现的功能是：通过 STM32 的 SPI 接口读写 FLASH。

参照串口下载将程序下载到开发板，就可以如下图 8- 7 界面：

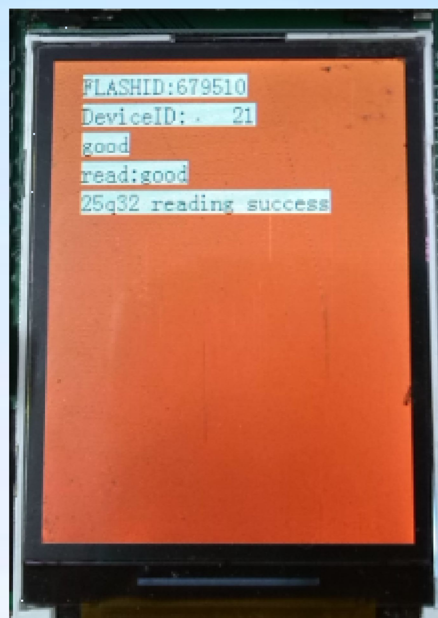


图 8- 7 读取 FLASH 显示图

表明读写 FLASH 成功。

STM32+LINUX 开发板预备篇

第 12 章 STM32+LINUX 开发板开发环境的构建

12.1 Windows 上虚拟机的安装及虚拟机上 Ubuntu 的安装

因为 Linux 是一种自由和开放源码的类 Unix 操作系统,所以我们的平台编程都是选择在 Linux 操作系统下进行的，而我们的电脑一般装载的都是 windows，因此我们需要在电脑上虚拟出另一个电脑。这样就可以在不改变我们本身的操作系统的同时，加入另外一种操作系统。在 Windows 安装 Linux 操作系统需要先在 Windows 上虚拟机的安装，然后在虚拟机上安装 Linux 操作系统。虚拟机安装方面，目前流行的虚拟机软件有 VMware(VMWare ACE）和 Virtual PC，它们都能在 Windows 系统上虚拟出多个计算机。在虚拟机上安装 Linux 操作系统方面，市场上 Linux 操作系统的版本很多，我们选择操作更方便，更易懂的 Ubuntu。

12.1.1 Windows 上虚拟机的识别

在安装之前我们需要先解压 VMware_Workstation_wmb，然后后进入该文件夹解压 VMware-workstation 注册机_keygen。准备工作完成后我们将正式介绍安装的整个过程。

双击 VMware-workstation-full-8.0.2-591240.exe，进入如图 1.1- 1 所示界面：



图 1.1- 1 虚拟机安装开始界面

出现如图 1.1- 2 所示的界面时点击“Next”：



图 1.1- 2 虚拟机安装欢迎界面

在图 1.1- 3 中，我们通常选择“Typical”模式：

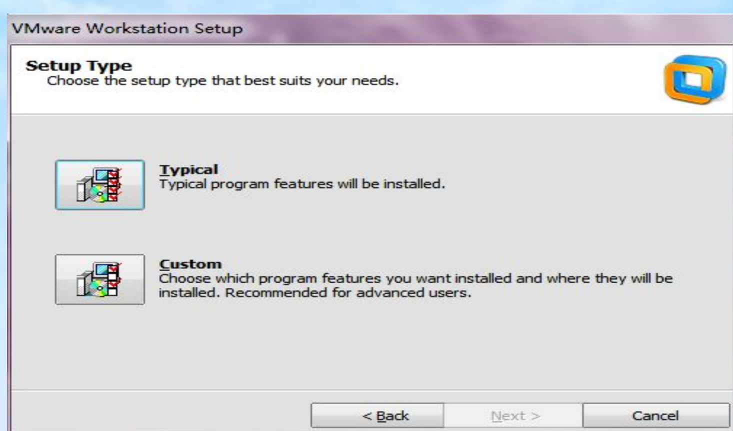


图 1.1- 3 虚拟机安装选择类型

接下来是选择安装路径，默认安装到C 盘，但因为软件过大建议更改安装路径，可点击“Change”，操作如图1.1- 4所示。我们选择安装在D盘的新建文件夹vm下，最后点击“Next”。

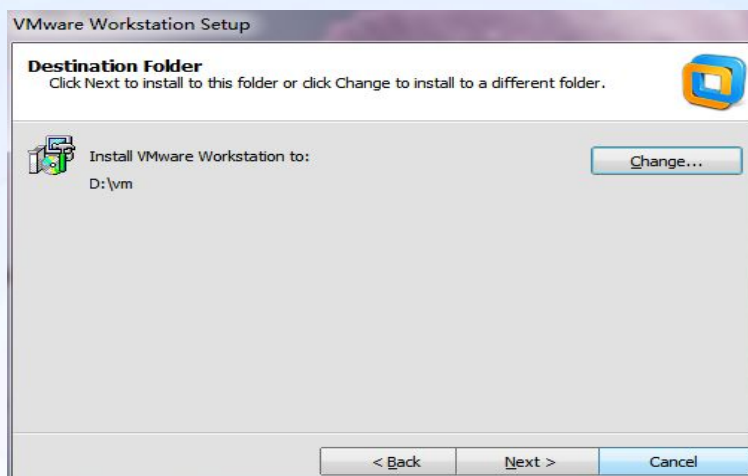


图 1.1- 4 选择安装路径

进入图 1.1- 5 所示界面，勾选 Check for product updates on startup 项（该选项为自选项），按“Next”

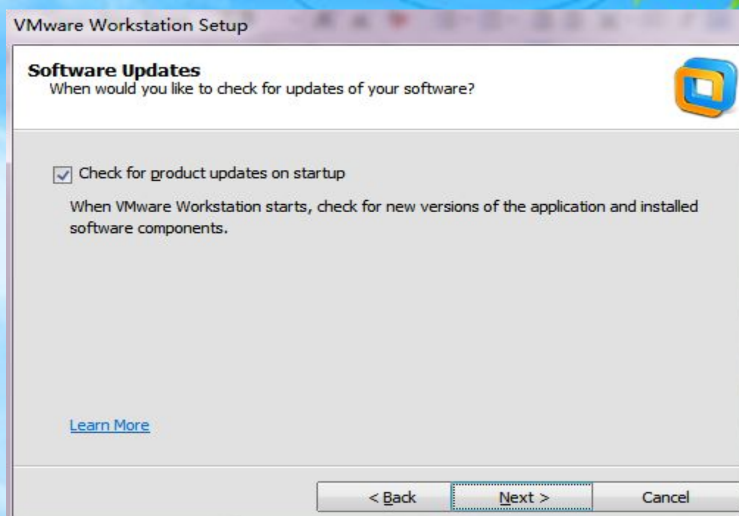


图 1.1- 5 选择是否检查更新

进入图 1.1- 6 所示界面，点击 Help improve VMware Workstation 选项(该选项为自选项)。按“Next”

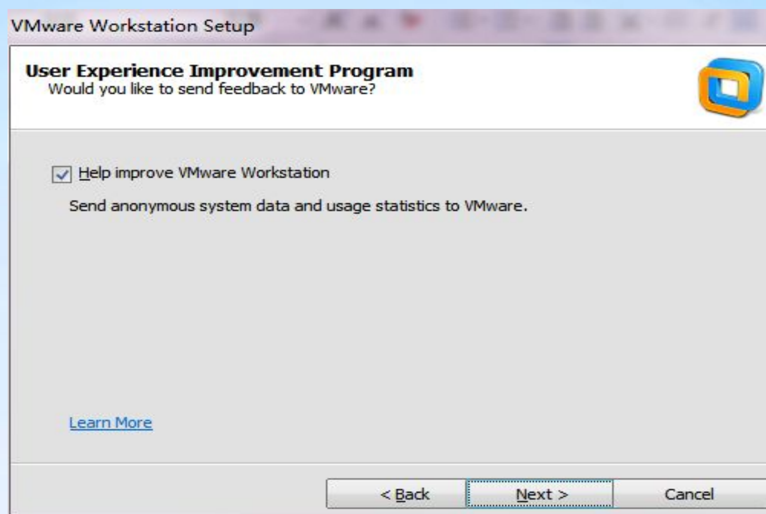


图 1.1- 6 选择是否帮助改进

进入图 1.1- 7 所示界面，勾选 Desktop 项和 Start Menu Programs folder 项。这两项都是自选项，建议都勾选，这样在桌面和 Windows 开始菜单项里都有 VM 的快捷方式。最后点击“Next”

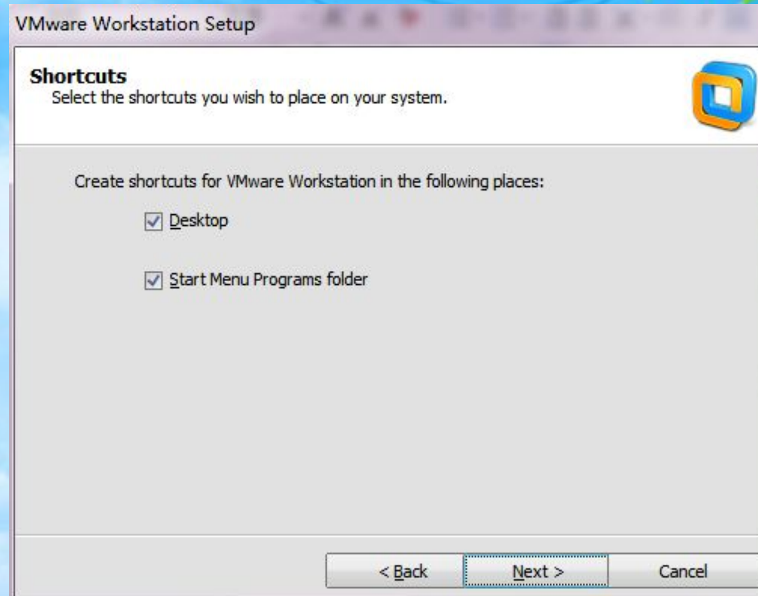


图 1.1- 7 是否创建快捷方式

进入图 1.1- 8 所示界面，点击“Continue”

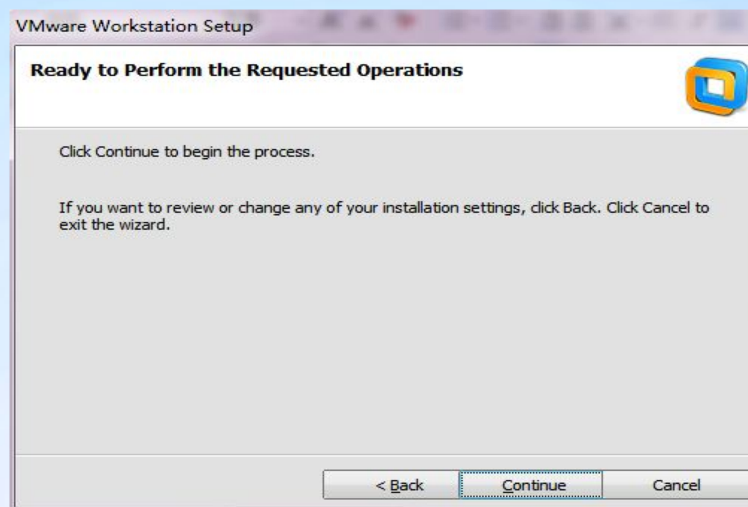


图 1.1- 8 准备安装

这时进入自动安装，如图 1.1- 9 所示。



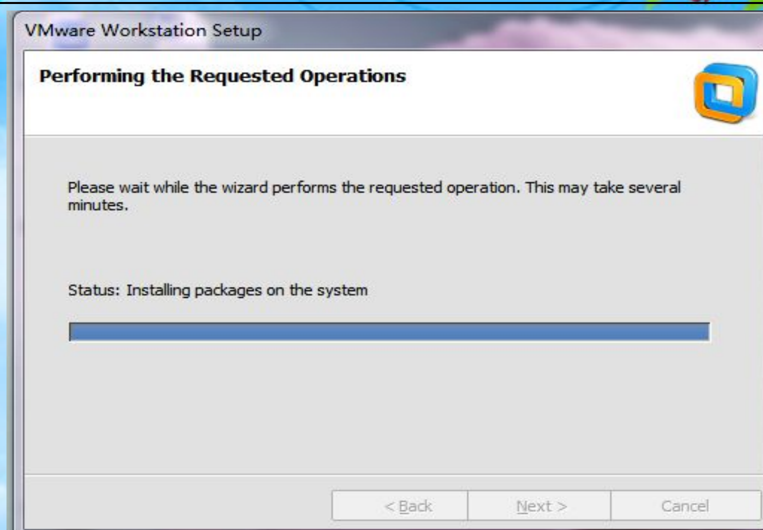


图 1.1- 9 安装界面

在自动安装完成之后出现如图 1.1- 10 所示界面。这时我们需要打开 VMware-workstation 注册机_keygen 文件夹下的有效注册密钥文件，将 UV3NK-25Y41-081CZ-0YP7C-PQ89A 复制在图 1.1- 10 中，然后按下“Enter”键

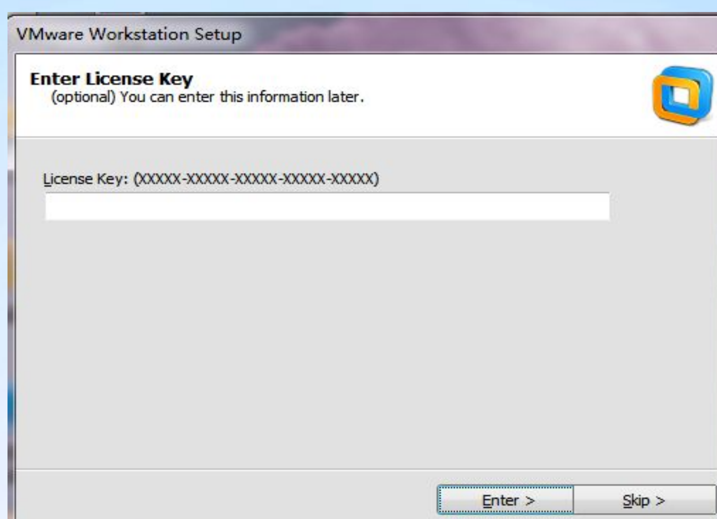


图 1.1- 10 注册界面

最后在图 1.1- 11 中点击“Finish”。

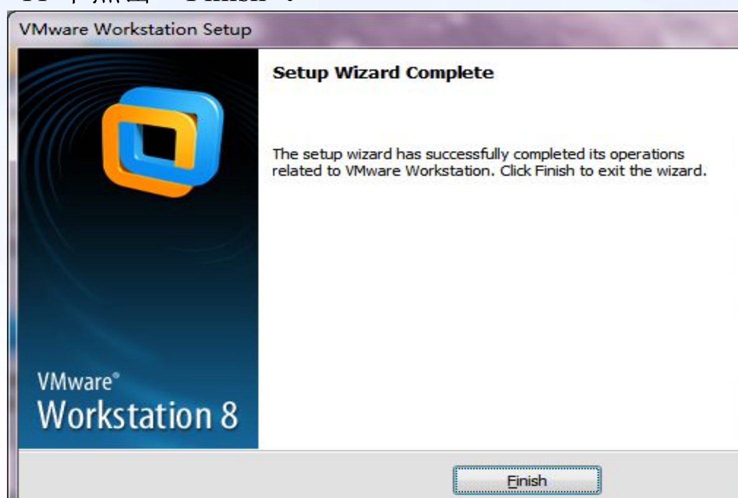


图 1.1- 11 虚拟机安装完成

到此，我们完成了在 Windows 下安装虚拟机的全部过程。接下来，我们介绍在虚拟机上安装 Ubuntu 的过程。

12.1.2 虚拟机上 Ubuntu 的安装

虚拟机上 Ubuntu 的安装需要自动下载一些文件，所以安装全程都需要使你的网络保持畅通。准备工作完成后，双击桌面的 VMware Workstation 的图标。
点击 Yes 项,后点 OK，如图 1.1- 12 所示：

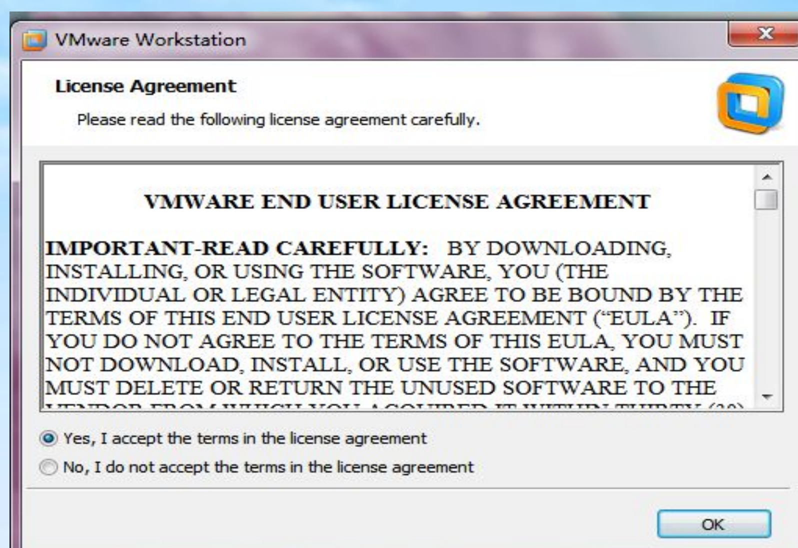


图 1.1- 12 虚拟机同意协议

点击 Create a New Virtual Machine 如图 1.1- 13 所示：

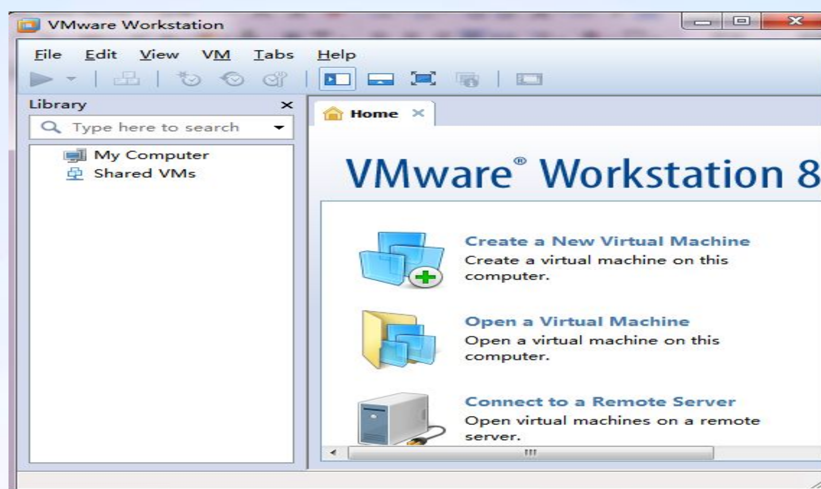


图 1.1- 13 虚拟机打开界面

这里我们通常选择 typical 选项。如图 1.1- 14 所示，点击 Typical，后 Next。



图 1.1- 14ubuntu 安装欢迎界面

点 击 Installer disc image file (iso) 选 项 。 点 击 browse ， 找 到 放 置 Ubuntu-11.10-desktop -i386.iso 的文件夹，后按打开项,最后按 Next 项，如图 1.1- 15 所示：

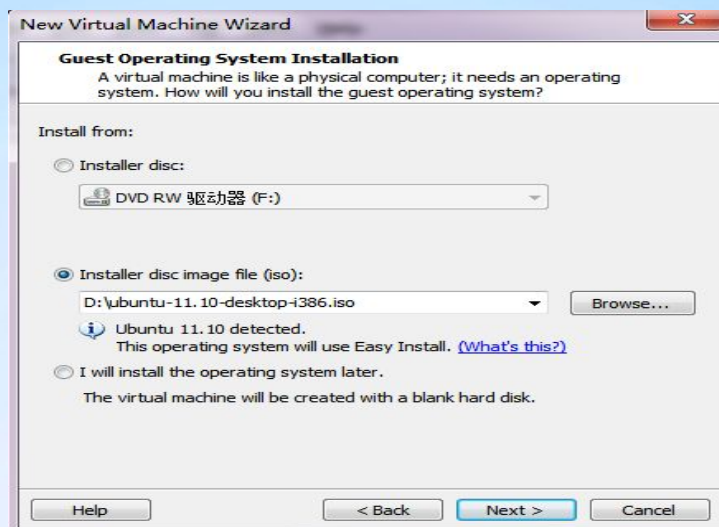


图 1.1- 15 选择安装镜像路径

完成以上配置后，出现如图 1.1- 16 所示。这些是你个人虚拟机一些开机信息的配置，有用户名和密码，可以根据个人喜好随意配置。如下下图配置，密码为 1，confirm 也为 1。



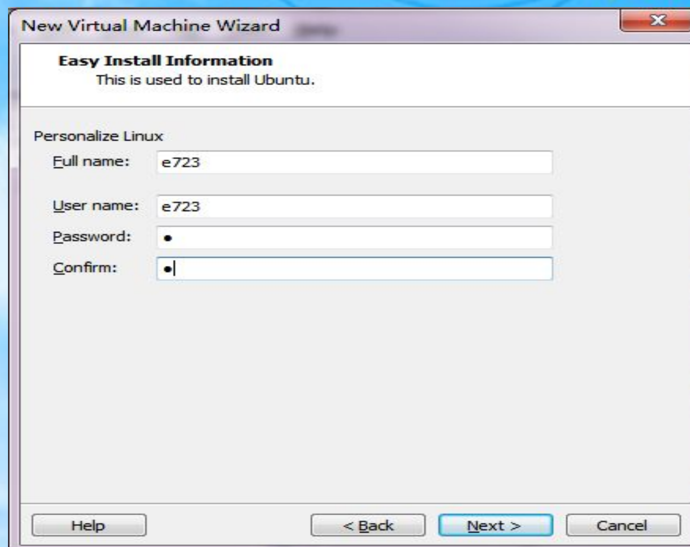


图 1.1- 16 输入用户名和密码

接下来是选择安装 Ubuntu 的路径，默认安装到 C 盘，但因为软件过大建议更改安装路径，可点击“Change”，操作如图 1.1- 17 所示。我们将 Ubuntu 安装到 D 盘新建文件夹 Ubuntu 内，最后按 Next

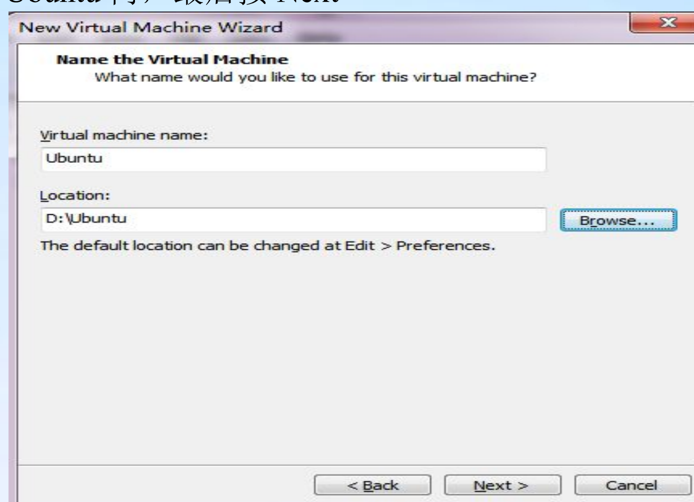


图 1.1- 17 选择 ubuntu 安装路径

Maximum disk size 选项可以为默认值也可以按如图 1.1- 18 所示的上下小按钮标志调节该大小值（这个根据个人的硬盘大小，适当调节），然后选择 Store virtual disk as a single file 项，最后点击 Next



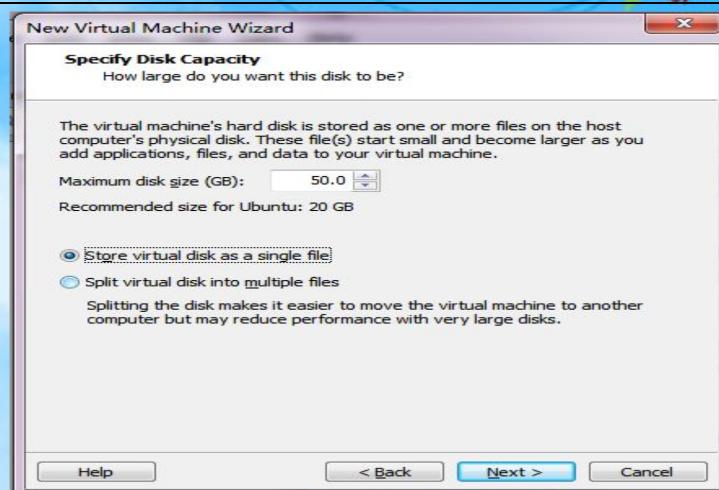


图 1.1- 18 设置硬盘大小

点击 Customize Hardware 项可以调节内存大小，处理位数等参数（也可以是默认值不需要调节）。点击 Finish，如图 1.1- 19 所示：

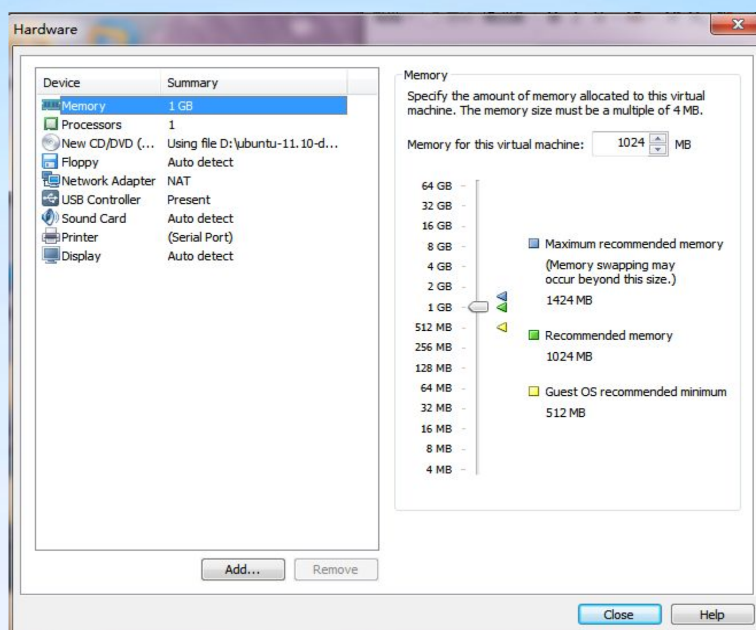


图 1.1- 19 设置内存等

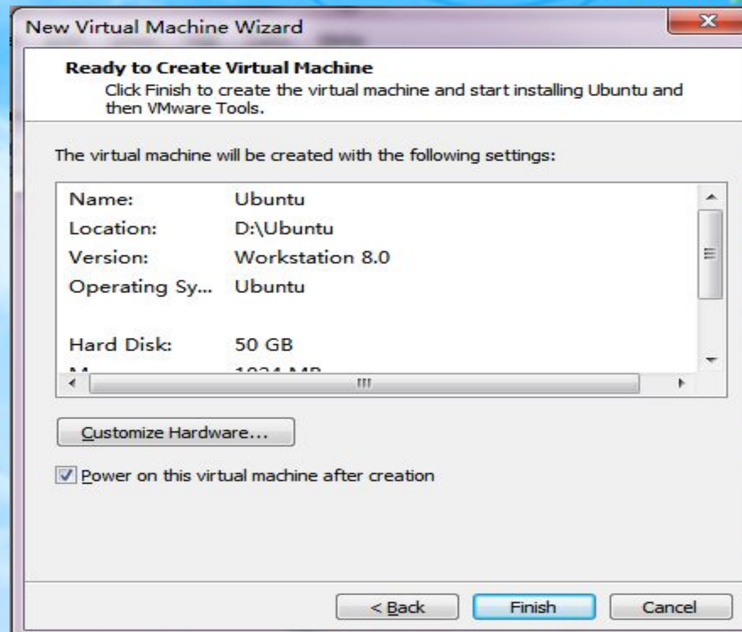


图 1.1- 20 显示配置信息

（提示：之后自动安装时出现的提示框都点击 OK 键。）

完成以上配置后这时就进入系统自动安装了，这时我们也需要进入漫长等待，一般要一个多小时。系统安装图如图 1.1- 21 所示。

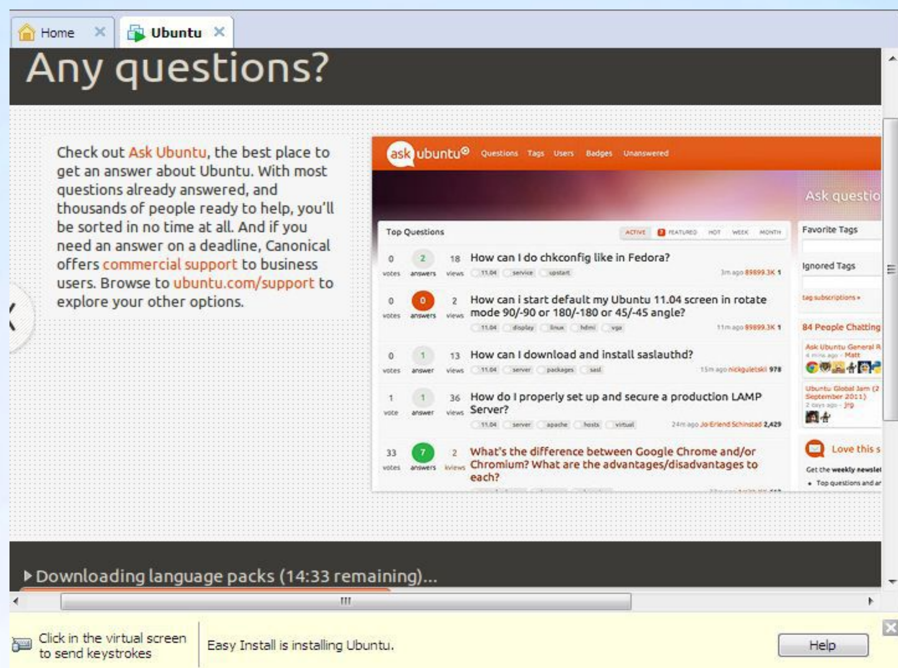


图 1.1- 21 正在安装

最后是是否删除 CD-ROM，点击 yes



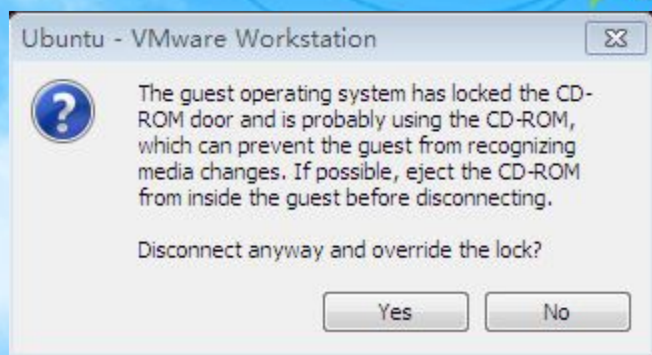


图 1.1- 22 是否删除 CD-ROM

12.2 Ubuntu 操作系统的开发环境介绍

12.2.1 Ubuntu 开发环境下的基本操作

我们对 Linux 系统用一些命令对其进行创建和修改。但是在哪输入这些命令呢？在 Dash 中搜索 terminal(终端),然后点击它的图标就可以了，或者用快捷键 Ctrl+alt+t 进入如下图的终端。所有的操作命令都是在如图 1.2- 1 终端的界面中输入。

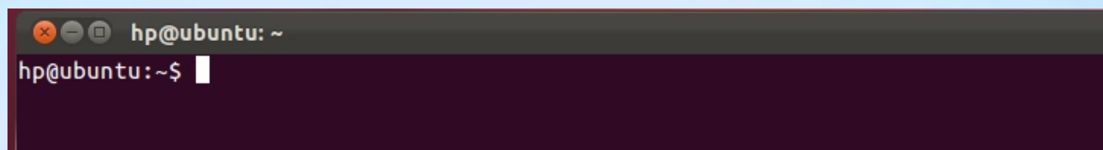


图 1.2- 1 终端显示界面

$ 为普通用户 按图 1.2- 2 操作可以切换为高级用户（也就是 root）

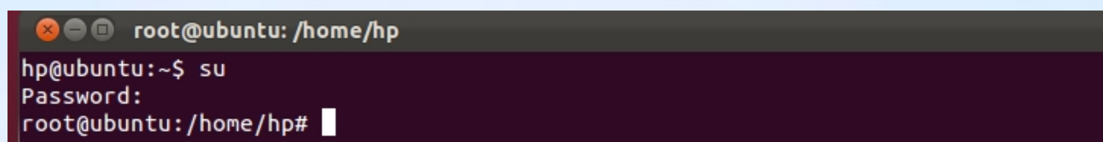


图 1.2- 2 切换为高级用户

（注意：根据版本的不同却换高级用户的命令还可以为 sudo -i）

12.3 嵌入式 Linux 中常用的命令

1. ls---查询命令

   查询当前目录下所有的文件及文件夹

2. cd---切换目录

   cd 命令的用法为：cd 具体的路径

   该命令能进入指定的路径，如 cd /etc 此时切换到 etc 文件夹下

   cd .. -->进入上一层目录

   cd / -->进入根目录

   cd -->进入用户主目录

cd -　　　　-->返回上一次所在目录

3. pwd ---显示当前所在的目录

4. mv---移动文件到指定的位置

mv 命令的用法为：mv 源文件 目标路径

该命令能将源文件移动到目标路径，如 mv /home/hp/*.conf　/home/weiyun

这是将 hp 目录下的*.conf，全部移到 weiyun 目录下。

5. cp---复制文件命令

cp 命令的用法为：cp　源文件 目标路径

该命令能将源文件复制到目标路径，如 mv /home/hp/*.conf　/home/weiyun

是将 hp 目录下*.conf 全部复制到 weiyun 目录下，源文件保留在 hp 文件夹内。

6. mkdir---新建文件夹

mkdir 命令用法为：mkdir　新文件夹名称

7. rmdir---删除文件夹

rmdir 命令用法为：rmdir 文件夹名称

8. cat---显示文本文件的内容

cat 命令的用法为：cat 文本文件名

该命令是查看文本文件内容，如 cat config.c，这时在终端会显示 config.c 的全部内容。

9. grep ---在某一个指定文本文件中查询指定的字符串

grep 命令用法为：grep 字符串 文本文件名。如 grep network　/etc/ntp.conf

10. touch ---新建文件

touch 命令用法为：touch　新文件名

11. chmod ---更改文件或文件夹权限

chmod 命令的用法：chmod　权限代码　文件名或目录名

该命令是修改文件名或目录名的权限问题。如 $ chmod　777　weiyun.c ,该命令是把 weiyun.c 这个文件改成可读可写。

12. uci ---设置有线或无线连接的相关选项

这里只介绍一下我们用到的相关命令。

设置 lan ip(即访问路由的 ip)命令：　uci set network.lan.ipaddr=[lan ip]

应用配置：uci commit network

12.3.1　嵌入式 Linux 下重要的热键

在我们的命令提示行模式里具有很多的功能组合键，这些按键可以辅助我们进行命令的编写与程序的中断。同时灵活的应用某些热键可以让我们在输入命令或较长的文件名称时更方便、更准确，可以避免很多输入错误的机会。在 Linux 下很常用的热键主要包括以下这些：

1. 采用上下光标键，使用以前所执行完成的命令：
2. 采用翻页键，使用以前所执行完成的命令：
   按下 pageup ，可定位到历史命令的第一条
3. 采用 Tab 键快速输入文件目录名：
   Tab 热键算是 Linux 的 Bash shell 最棒的功能之一了，它具有“命令补齐”



与“文件补齐”的功能。它可以有效避免我们打错命令或文件名，Tab 键几乎是使用最频繁的热键。用法为：在输入文件目录名之前,先输入一个或几个唯一的字符,再按 Tab 键，此时系统会自动将匹配的目录或文件名称补全。

4.采用“Ctrl+C”组合键结束当前进程：

如果你在 Linux 下面输入了错误的命令或参数，以至于导致程序会在系统下不停地运行，此时，我们按住“Ctrl+C”组合键就能让当前的程序“停下来”，它是中断目前程序的按键。

总之，在 Linux 下面，文字界面的功能是很强悍的，要多摸索、多用、多熟悉。

12.4 SecureCRT 软件介绍及应用

SecureCRT 支持多种协议，比如 SSH2、SSH1、Telnet、Serial 等。可以用它来连接 Linux 服务器，作为一个远程控制台进行各类操作；也可以用它来连接串口，操作目标板。这里我们主要是利用它的 SSH 功能，即用它来登陆开发板，作为终端来操作目标板，作用与 Linux 系统中的 terminal 终端的功能相近。

首先要保证电脑和开发板的网络畅通(可连接 SSID 名为 microcloud 的无线)

安装、启动 SecureCRT 后，跳过初始设置界面。点击菜单“文件”----“连接”，出现图 1.3- 1 所示界面，单击“新建会话”按钮，开始建立新的连接。

在随后出现的对话框中选择传输协议:这里我们选择串口协议 SSH,如图 1.3-2 所示。此时开发板是无线与 windows 相连的,。开发板与电脑之间通过网线连接的示意图如图 1.3- 3 所示:



图 1.3- 1 新建会话

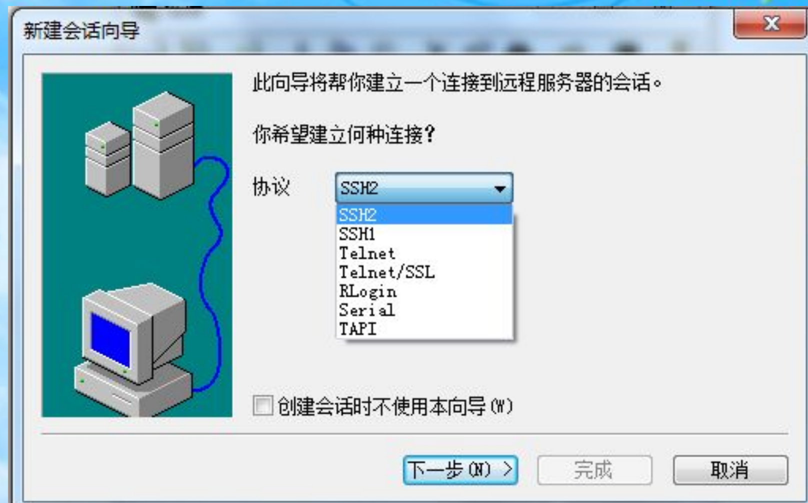


图 1.3- 2 选择协议

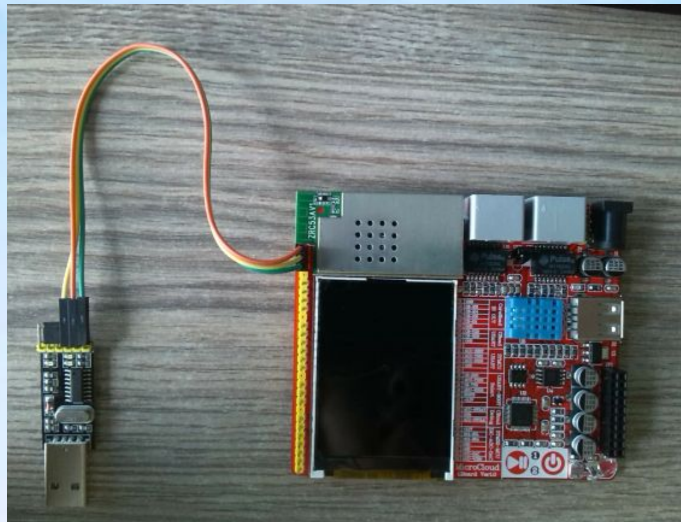


图 1.3- 3 串口线的连接

在图 1.3- 2 所示的对话框中单击“下一步”按钮，进行更详细的设置。对于 SSH 的设置请参考图 1.3- 4 进行设置。

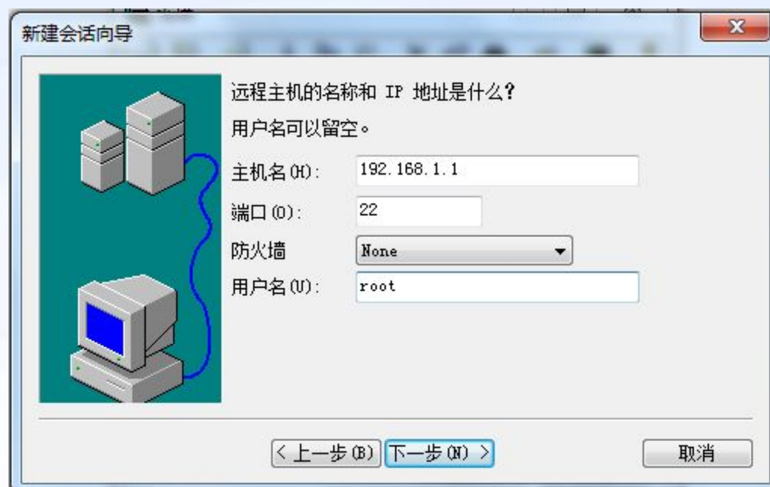


图 1.3- 4 配置串口参数

当建立完新的连接后，可以看到如图 1.3- 5 所示的对话框，也可以通过点击



菜单“文件”--“连接”启动这个对话框。在里面双击这个连接即可启动它。进行一次配置后，它会保存我们建立的会话，当我们再次用到此软件时，无需再进行配置(在连接的电脑串口保持不变的前提下)。直接双击会话即可进入如图 1.3-6 所示的界面：

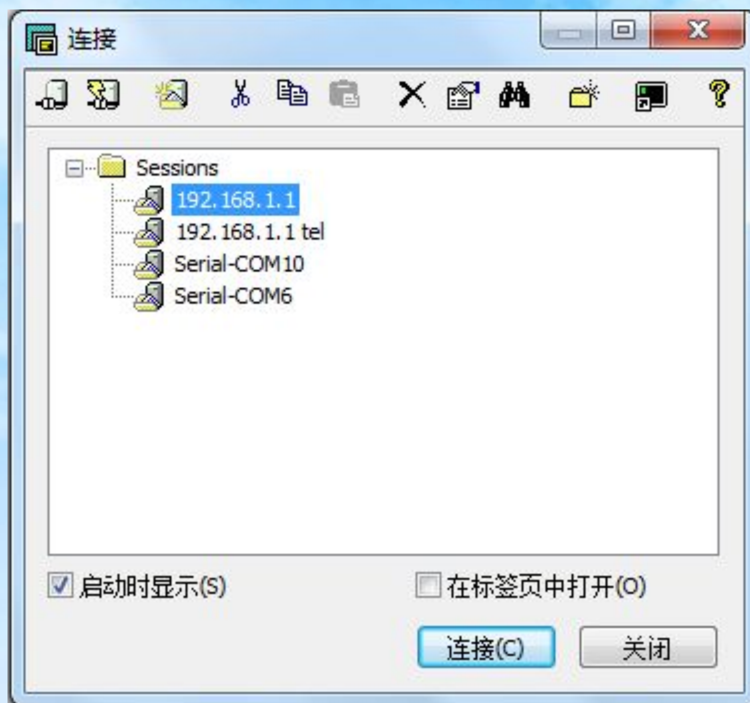


图 1.3- 5 会话配置完成

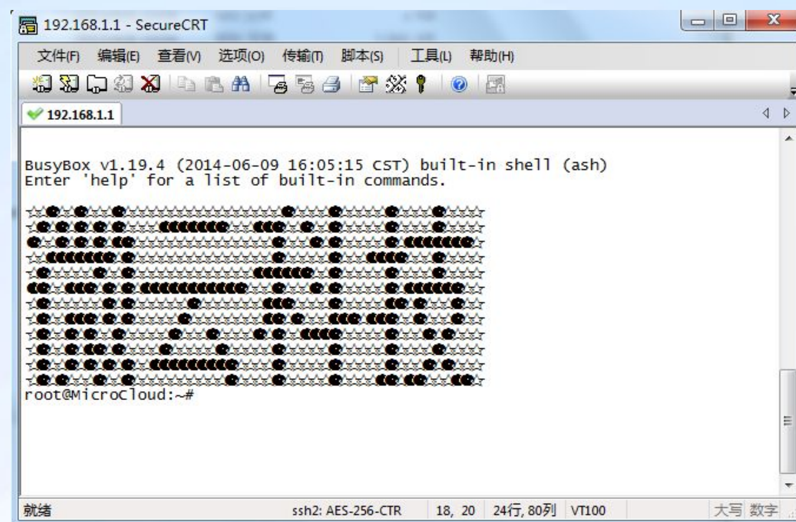


图 1.3- 6 会话连接成功

左上角 192.168.1.1 前面出现出现“对号”，说明连接成功！

开发板上电之后，等待大约 30s 会有开音乐响起来，然后我们点击连接上述会话，会有一个防火墙的提示信息，点确定，输入密码为 1，点击接受并保存（也可以点击只接受一次，但每次启动 SSH 都会有提示），出现下图的界面，就可以通过 SSH 控制开发板了。

```
BusyBox v1.19.4 (2013-10-10 20:21:12 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

root@MicroCloud:/#
root@MicroCloud:/#
```

图 1.3- 8 启动结束

“cd /”可进入根目录，使用 ls 命令查看根目录下的文件如图 1.3- 9 所示：

```
root@MicroCloud:/# ls
Program     etc          mnt       rom     sys     var
bin         lib          overlay   root    tmp     www
dev         lost+found   proc      sbin    usr
root@MicroCloud:/#
```

图 1.3- 9ls 命令

这里我们简单介绍一下根目录的意义以及根目录下各个子目录中应放置的文件内容：根目录是整个系统最重要的一个目录，因为不但所有的目录都是由根目录衍生出来的，同时根目录也与开机、还原、系统修复有关。由于系统开机时需要特定的开机软件、内核文件、开机所需程序、函数库等文件数据，若系统出现错误时，根目录必须要包含有能够修复文件系统的程序才行。如果以“账号”的角度来看，所谓的 root 指的是“系统管理员”的身份，如果以“目录”的角度来看，所谓的 root 指的就是根目录，就是/。根目录下各个子目录中存放的文件内容如表 1.3- 1 所示：

表 1.3- 1

目录	应放置文件内容
/Program	该文件夹用于暂时存放各种应用程序(包括可执行程序.o 文件，sh 脚本文件等)
/etc	系统主要的配置文件几乎都放在这个目录里，一般来说，此目录下的文件属性一般用户只可以查阅，只有 root 有权利修改。FHS 建议不要放置可执行文件(binary)在这个目录中。
/mnt	若想要暂时挂载某些额外的设备(如挂载 USB 设备)，一般建议可以放置到这个目录中。挂载成功后，会在 mnt 里出现相应设备的目录；
/rom	rom 是原 Flash 里的 root 目录。
/proc	这个目录本身是一个虚拟文件系统。它放置的数据都是在内存当中，例如系统内核、进程、外部设备的状态及网络状态等。
/sys	与/proc 类似，也是一个虚拟的文件系统，主要用于记录和内核相关的信息。包括目前已加载的内核模块与内核检测到的硬件设备信息等。
/var	此目录主要针对常态性变动的文件，包括缓存(cache)、登录文件(log file)以及某些软件运行所产生的文件。
/bin	此目录放置的是在单用户维护模式下还能够被操作的命令。在/bin 下面的命令可以被 root 与一般用户使用，主要有 cat、chmod、chown、date、mv、mkdir、cp、bash 等常用命令。
/lib	系统的函数库非常多，而/lib 放置的则是开机时会用到的函数库，以及在/bin 或/sbin 下面



	的命令会调用的函数而已。
/root	系统管理员(root)的主文件夹。
/tmp	这是让一般用户或者是正在执行的程序暂时放置文件的地方。此目录任何人均可访问，需要定时清理，所以重要数据不可放置在此目录下。
/www	此目录用于存放网页文件以及和网络相关的脚本文件等。
/dev	在 Linux 系统上，任何设备与接口设备都是以文件的形式存在于这个目录当中的。只要通过访问这个目录下面的文件，就等于访问某个设备。
/lost+found	此目录是使用标准的 ext2/ext3 文件系统格式才会产生的一个目录，目的在于当文件系统发生错误时，将一些丢失的片段放置到这个目录下。此目录通常会在分区的最顶层存在，例如你加装一块硬盘于/disk 中，此系统下就会自动产生一个这样的目录“/disk/lost+found”
/sbin	放在/sbin 下面的为开机过程中所需要的，里面包括了开机、修复、还原系统所需要的命令。
/usr	与软件的安装/执行有关的文件均放置在此目录。usr 为“UNIX software resource”的缩写。

12.5 WinSCP 软件介绍及应用

WinSCP 是一个在 windows 环境下使用的 SSH 的开源图形化的 SFTP 客户端，支持 ftp、SCP 协议。它的主要功能是本地与远程计算机的上传下载复制功能。也就是让我们的开发板与 windows 通信，可以直接的把文件在两者之间上传下载。

WinSCP 操作：

运行程序设置如图 1.4- 1 所示：

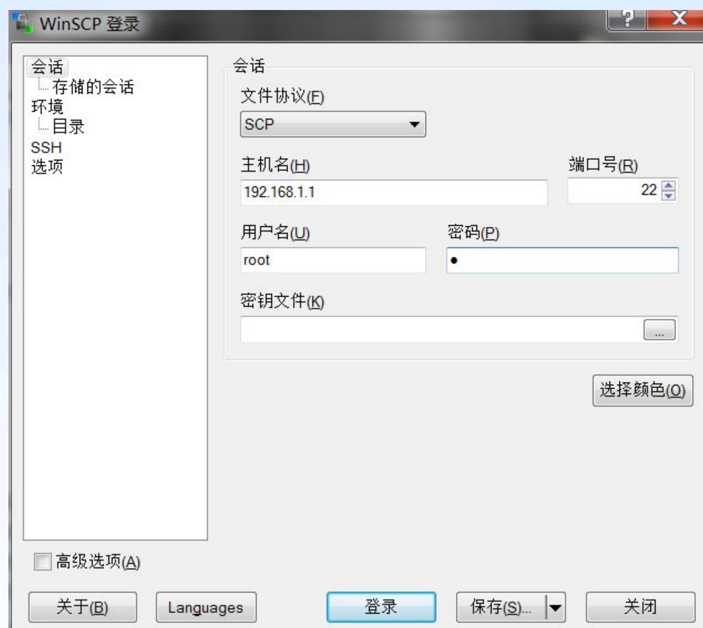


图 1.4- 1winSCP 设置界面

图 1.4- 1 为打开 WinSCP 的设置界面，这里的文件协议要选择 SCP，SCP 即“secure copy”是用来进行远程文件拷贝的。主机名是自己的下位机地址，即开发



板的 IP 地址 192.168.1.1。端口号选择 22，因为 SFTP 本身没有单独的守护进程，它必须使用 sshd 守护进程（端口号默认是 22）来完成相应的连接操作。开发板出厂时用户名为 root，密码为 1（STM32+linux 默认密码，可修改）。然后点击"保存"出现如图 1.4- 2 界面：

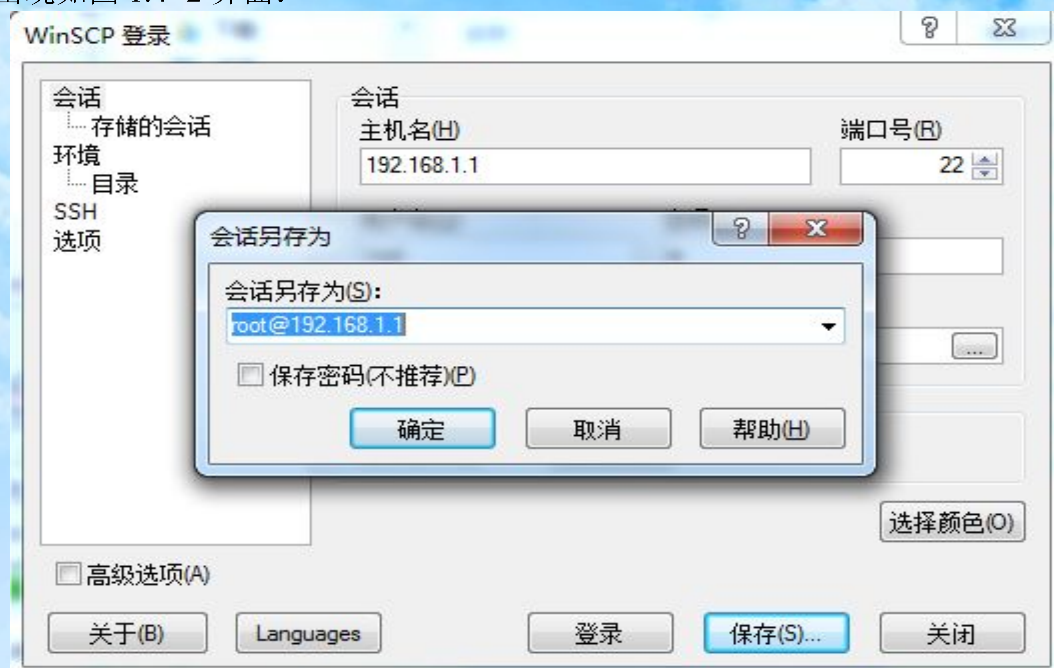


图 1.4- 2 保存会话

点击确定，这样我们下次登录时不用重新输入信息，在"存储的会话"中就会存在已经保存好的这个会话，选择它，点击"登录"即可，界面如图 1.4- 3：



图 1.4- 3 已保存的会话界面

登录过程中会出现如下两个错误提示：



图 1.4- 4 错误提示 1



图 1.4- 5 错误提示 2

这个错误提示是由于我们开发板不存在用户组、和其他用户，只有根目录。所以可以直接忽略掉它们，点击“确定”即可继续登录。

登陆后的界面如图 1.4- 6 所示。

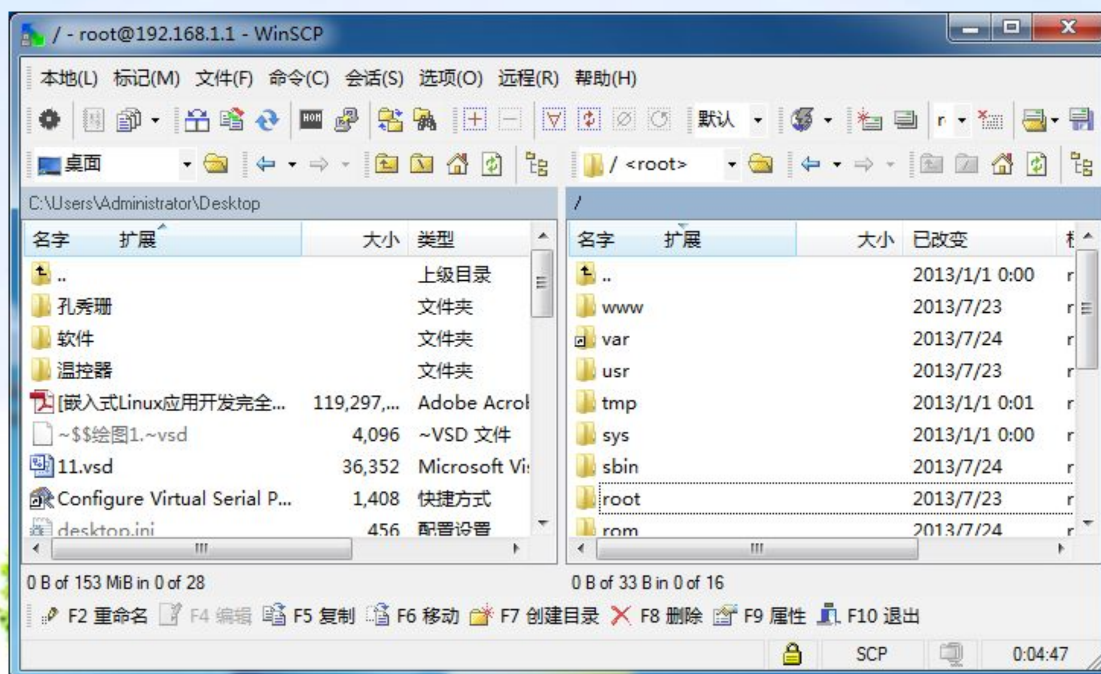


图 1.4- 6winSCP 主界面

左面是 windows 界面，右面是下位机，当我们想要把编译好的可执行文件传至下位机时，我们可以直接把需要的文件拖拽到下位机相应的文件夹内，例如我想把 windows 中的“UDP_send”文件复制到开发板中的“www”文件夹下，拖拽过程中会出现如图 1.4- 7 所示界面：



图 1.4- 7 复制文件

点击“复制”，这样我们就完成了上位机到开发板的文件传输。

12.6 Notepad++编辑软件的介绍及应用

Notepad++ 是一款 Windows 环境下免费开源的代码编辑器。除了可以用来制作一般的纯文字说明文件，也十分适合当作撰写电脑程序的编辑器。支持的语言: C, C++ , Java , C#, XML,Ada, HTML, PHP, ASP, AutoIt, 汇编, DOS 批处理, Caml, COBOL, Cmake, CSS, D, Diff, ActionScript, Fortran, Gui4Cli, HTML, Haskell, INNO, JSP, KIXtart, LISP, Lua, Make 处理(Makefile), Matlab, INI 文件, MS-DOS Style, NSIS, Normal text, Objective-C, Pascal, Javascript。用户可自定义编程语言：自定的编程语言不仅有语法高亮度显示功能，而且有语法折叠功能功能。注释、关键字和运算符号也可以自定义。是一款非常好用，而且功能强大的编辑软件，资料包里有这个软件的安装程序。安装完软件之后，打开界面如图 1.5- 1 所示：

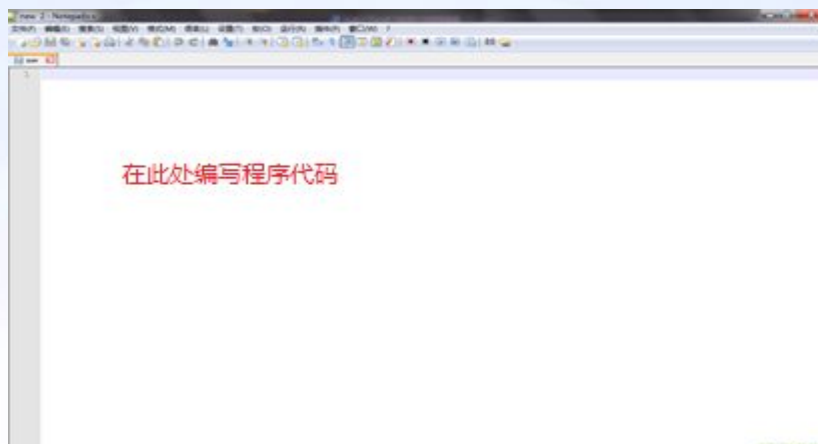


图 1.5- 1 编辑界面

比如我们编辑一个 shell（linux 下的一种编程）程序：

```
#! /bin/bash

echo "hello world!\n"
```

图 1.5- 2shell 脚本示例

先不去管程序的内容是什么，然后我们保存这个文件，点击菜单栏的文件，选择另存为，因为我们的是脚本文件，后缀名是以.sh 为结尾的，所以我们在保存类型里要选择 Unix script 一项：

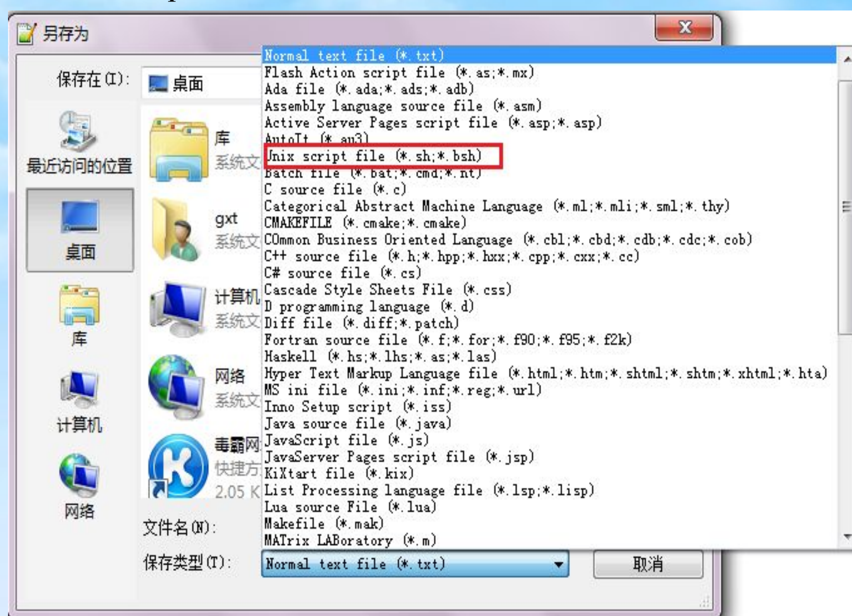


图 1.5- 3 保存文件

图 1.5- 3 可以看出，Notepad++可以编辑很多种类型的程序，使用起来非常的方便，这个也是本文经常用到的编辑和查看代码工具，因此在此处说明一下。

12.7 FileZilla Server Interface 软件介绍及应用

FileZilla 是 Windows 平台下一款不错的 FTP Server 软件，并且免费开源，分为服务器端和客户端，我们的开发板上配有 ftp 的客户端，所以上位机需要 FTP 服务器端，因此介绍一下 FileZilla Server 的安装与使用。

12.7.1 软件的开启

打开资料包中->所用到的工具->FileZilla_Server 文件夹，可以看到如图 1.6- 1 界面：

名称	修改日期	类型	大小
FileZilla Server Interface	2013/4/14 11:44	应用程序	1,006 KB
FileZilla Server Interface	2013/9/13 19:48	XML 文档	1 KB
FileZilla server	2013/4/14 11:28	应用程序	575 KB
FileZilla Server	2013/8/13 16:18	XML 文档	6 KB
FzGSS.dll	2012/2/26 22:42	应用程序扩展	81 KB
legal	2012/2/23 6:10	HTML 文档	2 KB
libeay32.dll	2012/2/26 22:50	应用程序扩展	1,085 KB
license	2011/11/6 20:27	文本文档	18 KB
readme	2012/2/26 22:41	HTML 文档	38 KB
ssleay32.dll	2012/2/26 22:50	应用程序扩展	270 KB
汉化说明	2013/4/14 11:54	文本文档	1 KB
河东下载	2013/5/21 19:27	Internet 快捷方式	1 KB
下载说明	2013/5/21 19:28	HTML 文档	4 KB

图 1.6- 1FileZilla 所在文件夹

双击第三个图标（运行服务后台程序），可以启动和关闭 ftp 服务（ftp 启动之后，进程在后台运行）

然后运行服务管理程序 File Sever Interface.exe.登陆服务管理程序的界面如图 1.6- 2 (管理密码可以不填)：



图 1.6- 2 登陆服务管理程序的界面

登陆进去以后的界面如图 1.6- 3：

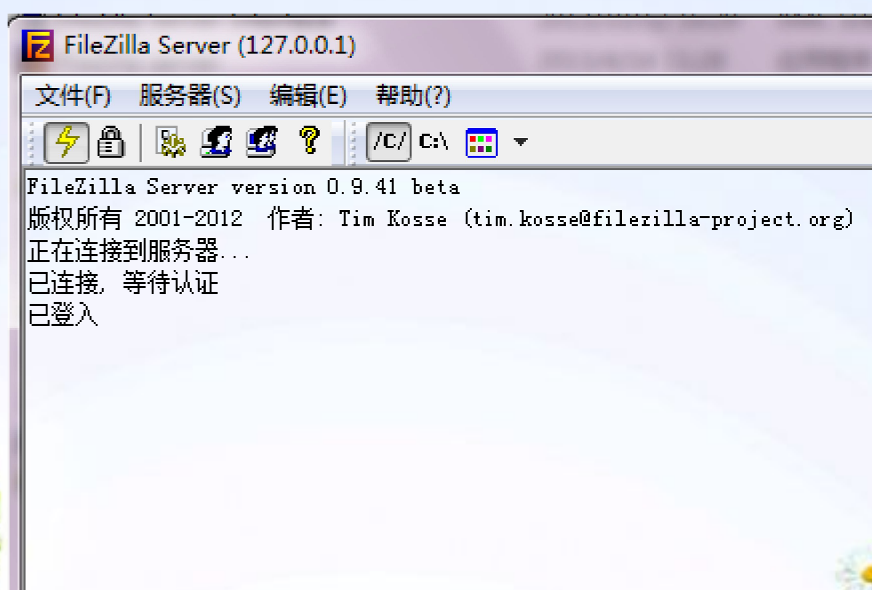


图 1.6- 3 登陆成功

12.7.2 软件的配置

在菜单栏中选择编辑->用户，弹出如图 1.6- 4 对话框：

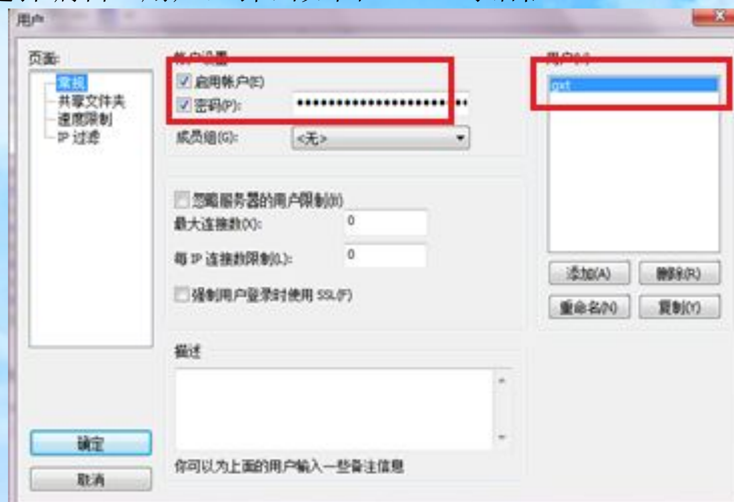


图 1.6- 4 配置用户密码

在右边添加或者删除用户名，这里我们只添加了一个用户名为“gxt”，在账户设置一栏将启用账户和密码勾选，然后设置密码，密码可随意设置，这里设为“1”。点击左侧的共享文件夹，设置共享的文件夹，如图 1.6- 5 所示：

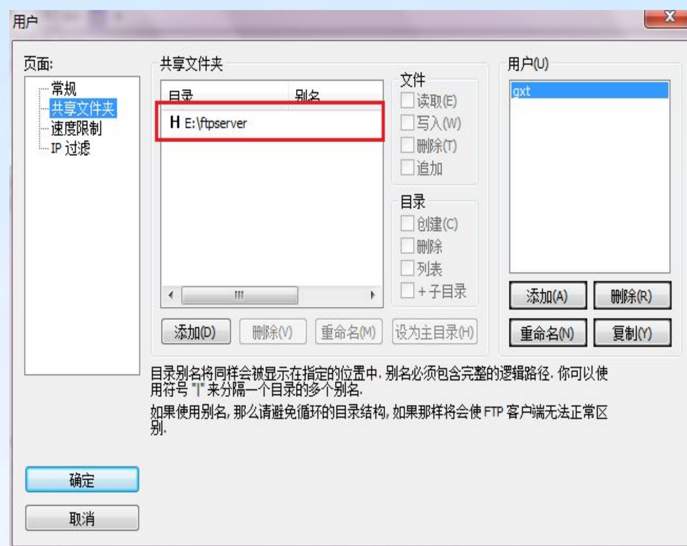


图 1.6- 5 设置共享文件夹

这里我们设置为 E:\ftpserver（新建的,“H+目录”）当其他主机和本机建立 ftp 连接之后，就可以查看这个目录下的文件了。

12.7.3 软件的验证

首先要知道本机的 IP，如果不知道本机的 IP 可在命令提示符下，用 ipconfig 查到，如图 1.6- 6 所示：

```
   媒体状态 . . . . . . . . . . . . : 媒体已断开
   连接特定的 DNS 后缀 . . . . . . . :

以太网适配器 本地连接:

   连接特定的 DNS 后缀 . . . . . . . : dns.hrbust.edu.cn
   本地链接 IPv6 地址. . . . . . . . : fe80::20cb:a520:1fd4:63a%10
   IPv4 地址 . . . . . . . . . . . . : 10.3.9.198
   子网掩码 . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : 10.3.9.254

隧道适配器 isatap.{D234C195-48DF-45A3-81C5-86080E656638}:

   媒体状态 . . . . . . . . . . . . : 媒体已断开
   连接特定的 DNS 后缀 . . . . . . . :

隧道适配器 Teredo Tunneling Pseudo-Interface:

   连接特定的 DNS 后缀 . . . . . . . :
   IPv6 地址 . . . . . . . . . . . . : 2001:0:5ef5:79fd:245e:3ac6:f5fc:f639
   本地链接 IPv6 地址. . . . . . . . : fe80::245e:3ac6:f5fc:f639%12
   默认网关. . . . . . . . . . . . . : ::
```

图 1.6- 6 查询本机 IP

我们这里的 IP 为 10.3.9.198.

打开浏览器，输入 ftp://本机 IP，即 ftp://10.3.9.198。弹出如图 1.6- 7 对话框：

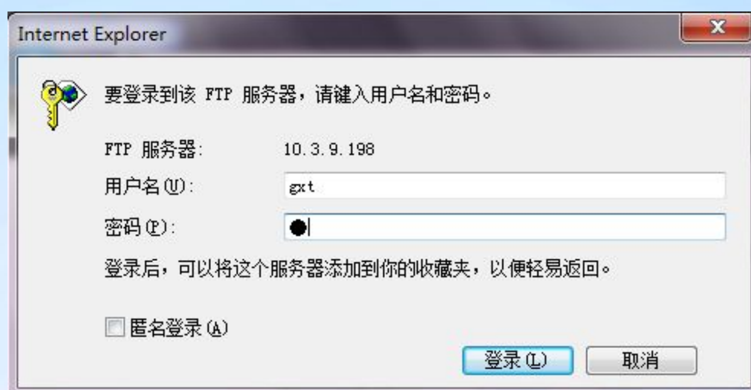


图 1.6- 7FTP 登陆界面

输入用户名和密码（用户名为 gxt，密码是 1），然后登陆，就可以在浏览器里查看到 E:\ftpserver 文件夹下的信息了。如图 1.6- 8 所示：



图 1.6- 8FTP 根位于本机

如果有两台 PC 机，并确保两个主机的网络畅通，一台机器开启 FileZilla_Server，在另一台机器的浏览器里也可以看到共享文件夹的内容。



STM32+LINUX 系统资源与功能篇

从上一部分的硬件讲解中我们知道，开发板上有两个主设备，中央模块和STM32 单片机，相信我们对单片机已经有了一定的基础了，所以我们对单片机只做简单的介绍。中央模块的功能主要有：

- 执行 C 语言应用程序。Linux 系统是基于 C 语言开发的，我们的 STM32+LINUX 开发板的中央模块是基于 Linux 内核搭建起来的，所以用 C 语言编辑的程序，编译成可执行程序，上传到我们的开发板上之后，也可以执行，从而实现我们的目标功能。
- 多种网络接入方式。开发板具备多种可以上网的方式：(1)运用一根网线可以将一台 PC 机和开发板相连，使二者可以进行网络通信；(2)可将开发板配置成 AP 模式，使其他的 PC 机通过无线，和开发板进行网络连接；(3)开发板可配置成 client 模式，即作为一台主机，接入其他 WIFI 接入点，从而接入广域网 Internet；(4)通过 3G 网卡接入广域网。
- lftp 上传下载文件。开发板支持 lftp 客户端上传文件功能，可实现主动向 ftp 服务端拨号，上传下载文件功能。
- 驱动摄像头功能。中央模块的系统里配有摄像头的驱动程序，当摄像头的驱动开启之后，可从浏览器里远程的监控摄像头的画面。
- 浏览和查看网页信息。因为开发板是一个小型的 web 服务器，所以只要将编写好的网页文件上传到开发板的相应文件夹(www 文件夹)下，在其它与开发板网络通信正常的主机浏览器里，就可以浏览到开发板上保存的网页信息。
- 发送和接收邮件功能。学习板配有邮件的收发功能，同时也可以具有用手机短信回复邮件的功能，以此功能，我们可以用短信的方式远程的控制开发板上的外围设备，比如 led 的亮灭，红外遥控等。
- Samba 共享文件夹。Samba 共享可实现在任意一台与开发板联网的主机上，查看开发板上的文件信息，方便查询和管理。
- 流媒体播放。开发板支持 USB 声卡，支持 MP3 格式的音频播放。

第 13 章 采集与控制部分的资源与功能

13.1 数据采集与控制程序的仿真与下载

开发板的单片机部分主要包括 LCD 液晶彩屏通过读取 Flash 存储芯片显像、触摸按键、DHT11 数字温湿度传感器以及红外对管四部分。

单片机程序下载方式有两种，串口下载和 ST-LINK 下载。串口下载参考单片机详解篇的串口下载章节。

13.2 2.8 寸静态显示液晶屏功能

图 2.2- 1 为 LCD 液晶彩屏和 Flash 存储芯片实物图，LCD 液晶彩屏要显示的图片数据已经事先烧录到 Flash 存储芯片中。在 STM32+linux 系统中，LCD 液晶彩屏显示图像的主要原理是：单片机读取 Flash 中存储的图像数据，然后将图像数据逐一写进 LCD 液晶屏进行图像显示。

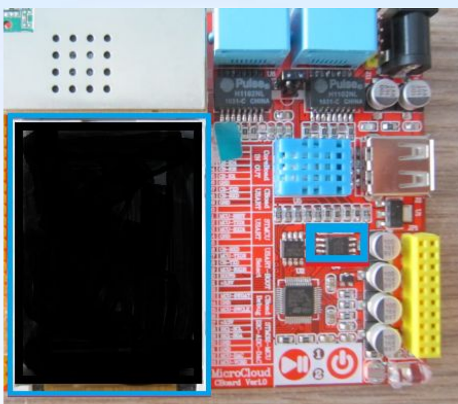


图 2.2- 1 液晶屏和 Flash 存储芯片实物图

STM32+LINUX 开发板中采用的 LCD 液晶彩屏为 2.4 寸 TFT LCD，并且是由 320*240 个像素点组成显像。为 LCD 液晶彩屏硬件原理图，各个引脚与图 2.2-2 中的 STM32 单片机红框中的各个管脚相连接。

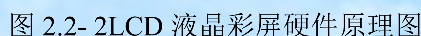

图 2.2- 2LCD 液晶彩屏硬件原理图


图 2.2- 3 STM32 单片机

13.3 大数据文件存储 Flash 存储芯片

图 2.3- 1 为 STM32+LINUX 开发板中的 Flash 存储芯片硬件图，采用的 Flash 存储芯片的存储容量是 4M 字节。由于写保护引脚 WP 可以被用来保护状态寄存器不被意外改写，写保护引脚接 3.3V。保持引脚 HOLD 拉高器件正常工作，所以 HOLD 引脚接 3.3V。


图 2.3- 1Flash 存储芯片硬件原理图

Flash 存储芯片与 STM32 单片机的引脚连接图如图 2.3- 2。其中，串行时钟引脚 FLASH-CLK 为输入输出提供时序。芯片选择引脚/CS 为高电平时，Flash 存储芯片会被禁能，DO 引脚高阻抗。此时，如果 Flash 存储芯片没有执行擦除指令、编程指令或处于状态寄存器周期进程，Flash 存储芯片将处于待机阶段。芯片选择引脚/CS 为低电平时，使能芯片，此时可以进行芯片的读写操作。所以上电后，执行一条新指令前必须使/CS 引脚产生一个下降沿。

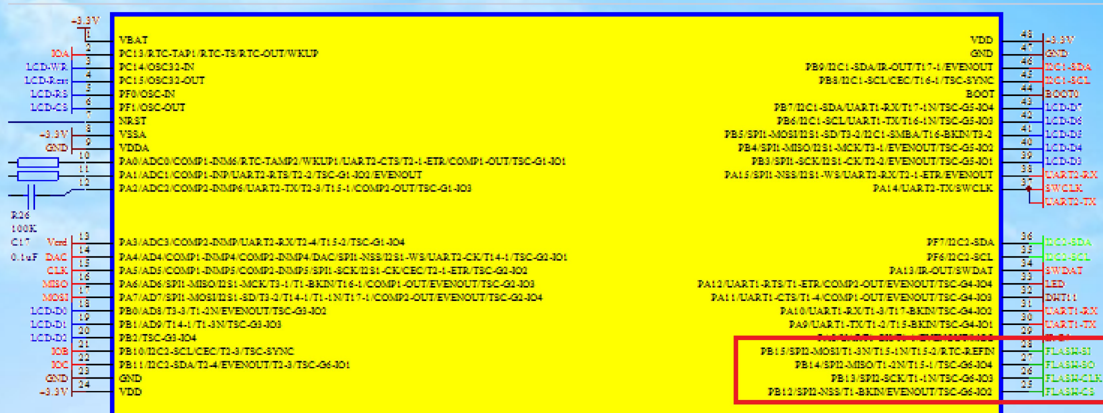


图 2.3- 2STM32 单片机引脚连接图

13.4 温湿度传感器

DHT11数字温湿度传感器因其具有数字模块采集技术和温湿度传感技术，被应用于家电、医疗、汽车以及气象站等领域。在图2.4- 1温湿度传感器实物图中，



图 2.4- 1 温湿度传感器实物图

图2.4- 2为DHT11数字温湿度传感器硬件图。DHT11数字温湿度传感器硬件图中NC脚悬空。SDA为数据传输线，DHT11根据高低电平的时间的长短，从SDA引脚读取温湿度的数据。

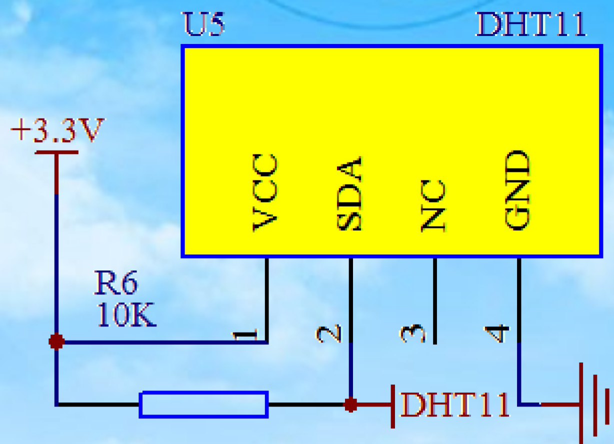


图 2.4- 2DHT11 数字温湿度传感器硬件图

STM32 单片机中与 DHT11 相连接的引脚是 PA11 脚，我们只需操作 PA11 口即可达到操作 DHT11 的目的。DHT11 的工程此处不做详细讲解。

13.5 红外控制中心之红外发射管组

红外通讯技术利用红外线来传递数据，属于无线通讯技术的一种。由于，红外通讯具有体积小、功耗低、功能强等特点，被广泛用于空调机、音响设备、录像机等装置上。在 STM32+LINUX 开发板中，采用如图 2.5- 1 所示的红外发射管来发射红外信号。

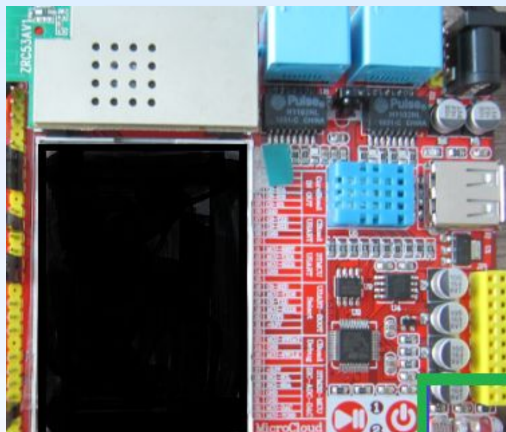


图 2.5- 1 红外发射管实物图

如图 2.5- 2 红外发射硬件电路中，数据从 IR_OUT 端输入，形成红外发射信号，再通过红外发射管将此电信号转化为光信号发射出去。

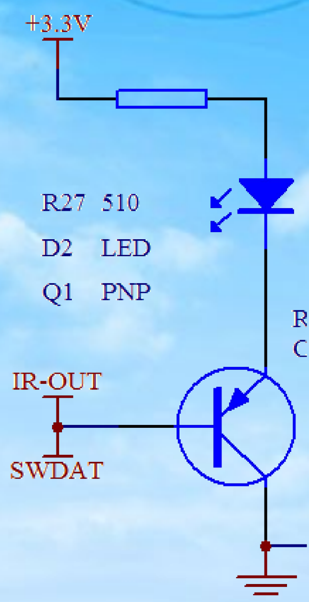


图 2.5- 2 红外发射硬件电路图

第 14 章 嵌入式 Linux 系统编程准备

我们都知道，在 windows 下，C 程序是在某种编辑软件里编写之后，编译成可执行文件，然后点击运行，C 程序就开始执行了。对于嵌入式开发，需要我们用主机为它构建基本的软件系统，然后烧写到设备中。比如单片机实际上就是在 PC 机上编写 C 语言程序，用专门的软件编译连接生成可执行文件，再用烧录工具烧写进单片机，程序就运行起来了。我们的 STM32+LINUX 开发板也是嵌入式设备，不能够实现直接在开发板上编译 C 程序，所以需要在主机平台上安装对应的交叉编译工具链（cross compilation tool chain），然后用这个交叉编译工具链编译我们的源代码，最终生成可在目标平台上运行的代码，然后移入我们的开发板上运行，这种“在一个平台上生成另一个平台上的可执行代码”的过程我们成之为交叉编译。

14.1 嵌入式 Linux 系统 C 语言源程序编写

我们先以最基本 helloworld 程序为例。

打开虚拟机，启动我们的 Ubuntu 操作系统，在某个文件夹（笔者习惯在主文件夹）下新建一个空白文件(右击鼠标->创建新文档->空白文档)，命名为 helloworld.c，如图 3.1- 1 所示：



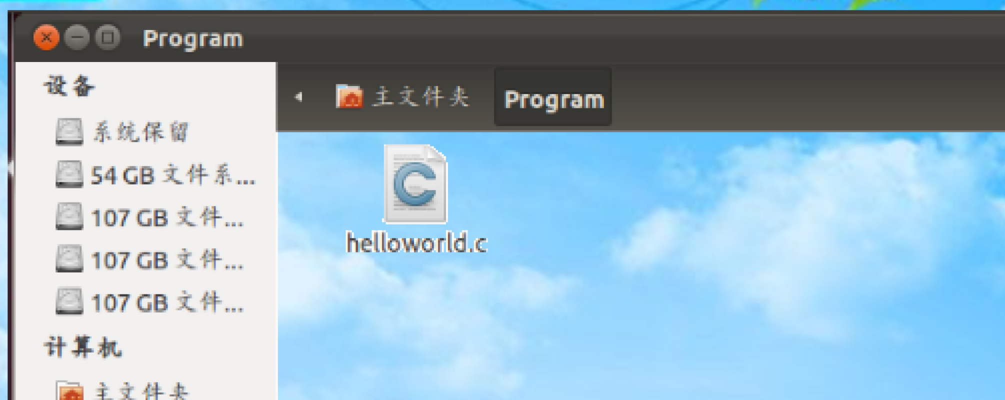


图 3.1- 1helloworld.c

打开这个文件，输入内容如图 3.1- 2 所示：

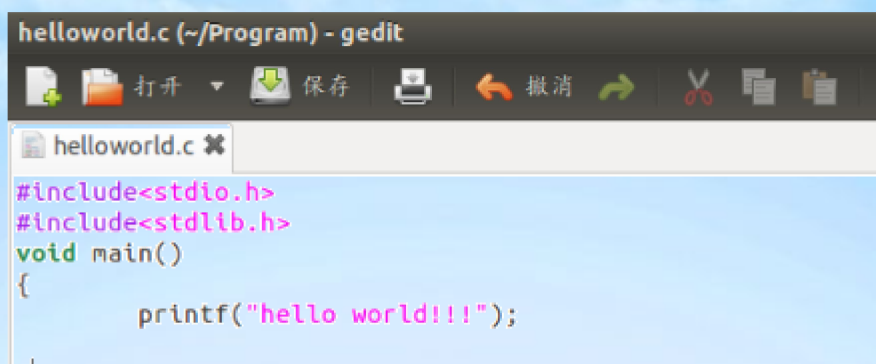


图 3.1- 2helloworld.c

该程序的作用就是在终端输出一句“hello world!!!”。

14.2 嵌入式 Linux 系统编程之编译与运行

在 windows 下，我们对程序的编译是直接点击编译器的某个按钮就完成了编译或者链接的过程（比如编写单片机程序的 keil），而在 Linux 下，编译和连接的过程是通过命令来实现的。Linux 系统下的 gcc 是最常用的编译器，一般的 Linux 系统都装有这个强大的编译器，它的基本用法如下：

语法：gcc [选项] 文件列表

说明：运行 gcc 是将完成预处理、编译、汇编、和连接四个步骤，并最终生成可执行文件，这个可执行文件默认保存为 a.out。gcc 可以接受多种类型的文件，也有超过 100 个的编译选项可以用，而我们最常用的，就是-o，即定制可执行文件的名称。以上面的程序为例我们介绍一下在 Linux 下编译的过程。

1）程序的编译

首先启动终端（可以按 Ctrl+Alt+T）如图 3.2- 1 所示：

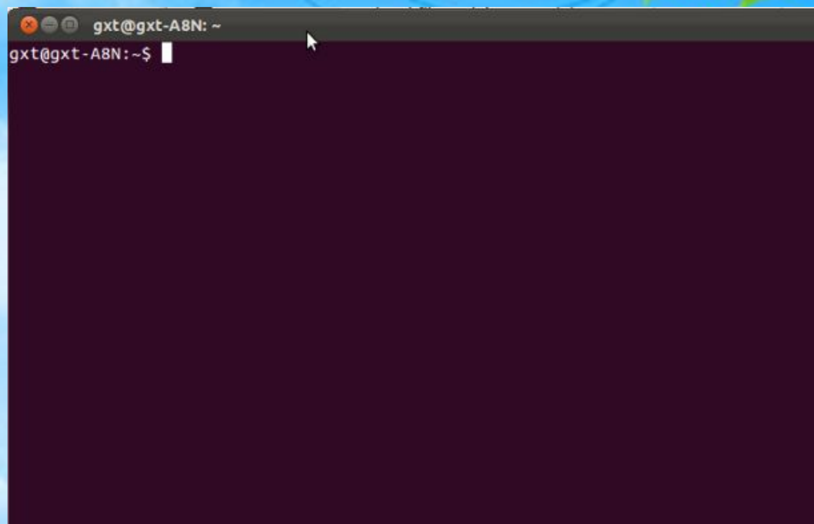


图 3.2- 1 启动终端

现在的状态是在主文件夹下，我们进入我们存放程序的文件夹下：

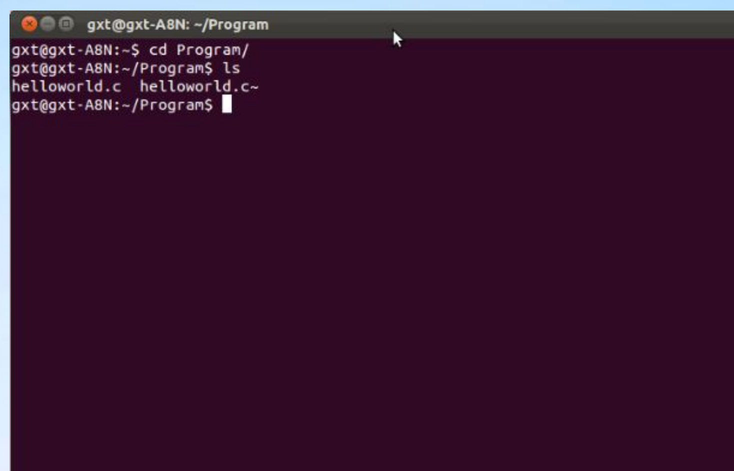


图 3.2- 2ls 命令

执行 ls 命令之后我们可以看到当前文件夹下含有我们的 helloworld.c 的文件（带有~标识表示文件正在打开或占用），输入命令：

gcc -o helloworld.o helloworld.c

命令是将 helloworld.c 文件编译成可执行文件，并将可执行文件命名为 helloworld.o，这个名字可以是任意命名的。命令执行之后，如果程序有问题，则在下面会有提示，如果程序没有问题，则在当前目录下会生成一个 helloworld.o 的文件，如图 3.2- 3 所示：

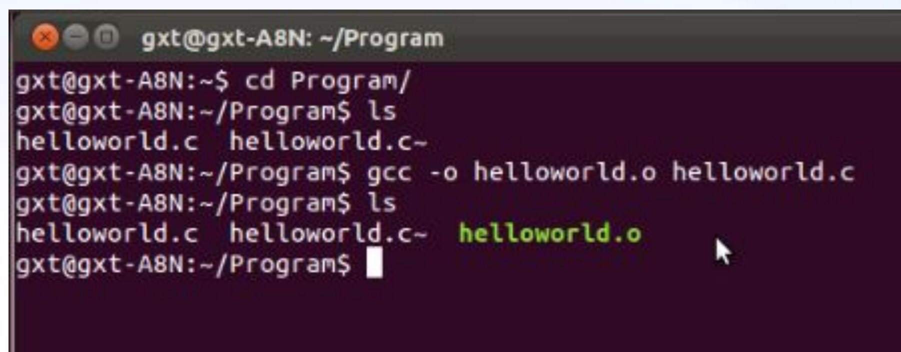


图 3.2- 3 显示 helloworld.c 程序

2）程序的运行

在终端里输入命令./helloworld.o 即可运行程序，我们可以得到如图 3.2- 4 结果：

```
gxt@gxt-A8N: ~/Program
gxt@gxt-A8N:~$ cd Program/
gxt@gxt-A8N:~/Program$ ls
helloworld.c  helloworld.c~
gxt@gxt-A8N:~/Program$ gcc -o helloworld.o helloworld.c
gxt@gxt-A8N:~/Program$ ls
helloworld.c  helloworld.c~  helloworld.o
gxt@gxt-A8N:~/Program$ ./helloworld.o
hello world!!!gxt@gxt-A8N:~/Program$
```

图 3.2- 4 运行结果

可以看到显示我们想要显示的字符串了。因为我们在当前目录下执行可执行文件，所以./helloworld.o 用到的是相对路径，若是在别的文件夹内执行的话，可采用绝对路径的方式：./home/gxt/Program/helloworld.o，也是可以运行程序的。

14.3 程序的交叉编译和在开发板上的运行

上一小节中在 Linux 环境下编译出的可执行文件 helloworld.o，只能在当前的 Linux 环境下运行，不能在我们的开发板上运行，所以我们用我专有的工具链进行交叉编译，编译出能在我们开发板上运行的可执行文件。该工具链在我们的资料包中“所用到的工具”文件夹下，是 OpenWrt-Toolchain-ramips-for-mipsel_dsp-gcc-4.6-linaro_uClibc-0.9.33.2.tar.bz2 的压缩包，我们将其拖拽进我们的 Ubuntu 系统解压后，存放到一个目录下，我们这里存放的目录是直接存放到了主文件夹下，并重新命名为 toolchain，如图 3.3- 1 所示：

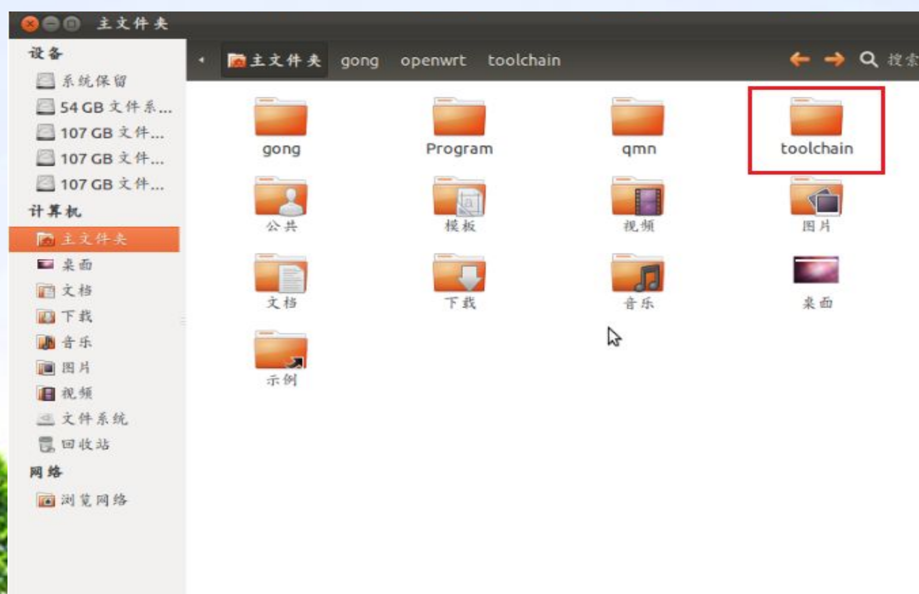


图 3.3- 1 工具链所在路径

1）程序的交叉编译

我们在终端输入命令：/home/gxt/toolchain/toolchain/bin/mipsel-openwrt-linux-gcc -o hello_world.o hello_world.c。如图 3.3- 2 所示：

```
gxt@gxt-A8N: ~/Program
gxt@gxt-A8N:~$ cd Program/
gxt@gxt-A8N:~/Program$ ls
helloworld.c  helloworld.c~
gxt@gxt-A8N:~/Program$ gcc -o helloworld.o helloworld.c
gxt@gxt-A8N:~/Program$ ls
helloworld.c  helloworld.c~  helloworld.o
gxt@gxt-A8N:~/Program$ ./helloworld.o
hello world!!gxt@gxt-A
gxt@gxt-A8N:~/Program$
gxt@gxt-A8N:~/Program$ /home/gxt/toolchain/bin/mipsel-openwrt-linux-gcc -o hello
world.o helloworld.c
mipsel-openwrt-linux-uclibc-gcc.bin: warning: environment variable 'STAGING_DIR'
 not defined
mipsel-openwrt-linux-uclibc-gcc.bin: warning: environment variable 'STAGING_DIR'
 not defined
mipsel-openwrt-linux-uclibc-gcc.bin: warning: environment variable 'STAGING_DIR'
 not defined
gxt@gxt-A8N:~/Program$
```

图 3.3- 2 交叉编译

出现三个警告，这个可以忽略。/home/gxt/toolchain/bin/mipsel-openwrt-linux-gcc 是指交叉编译工具 mipsel-openwrt-linux-gcc 存放位置的绝对路径，其中 gxt 指的是 Linux 主机的用户名，这一项是用户安装 Linux 系统时自定义的用户名。编译完之后，在当前文件夹下出现的 helloworld.o 就是我们新编译出的可执行文件，它只能在我们的开发板上跑，而不能在 Linux 系统下跑了。

2）程序的运行

用网线将电脑与开发板的网口连接之后（目的是保证 PC 机和开发板保持网络畅通，以便可以用 WinSCP 上传文件）给我们的开发板上电，在 SecureCRT 里可以看到启动信息，启动完成之后，打开 WinSCP 上传文件工具，将我们编译的 helloworld.o 文件上传到开发板中，这里我们新建了一个 Program 的文件夹，把文件上传到这个文件夹中。

上传之后可以在 SecureCRT 软件中查看到的文件：

```
root@MicroCloud:/# ls
Program  dev  lib  overlay  rom   sbin  tmp  var
bin      etc  mnt  proc     root  sys   usr  www
root@MicroCloud:/# cd Program/
root@MicroCloud:/Program# ls
helloworld.o
root@MicroCloud:/Program#
```

图 3.3- 3 查看可执行文件

如上图所示，我们可以看到 hello_world.o 可执行文件已经被上传到开发板中了。但是由 WinSCP 上传到开发板中的文件一般默认是不可执行的，所以输入执行的命令后会提示以下信息：

```
root@MicroCloud:/Program# ls
helloworld.o
root@MicroCloud:/Program# ./helloworld.o
/bin/ash: ./helloworld.o: Permission denied
root@MicroCloud:/Program#
```

图 3.3- 4 没有权限

执行之后提示有错误。因为我们没有权限来执行此文件，需要改变文件的权限。输入 chmod 777 hello_world.o，使其变为可读可写可执行的文件，

```
root@MicroCloud:/Program# chmod 777 helloworld.o
root@MicroCloud:/Program# ./helloworld.o
hello world!!!root@MicroCloud:/Program#
```

图 3.3- 5 执行成功

然后就运行，就可以看到我们想要的“hello world!!!”的字样了。

第 15 章 STM32+Linux 系统多种广域网接入模式

本章节主要从两方面来介绍开发板通过网络与外界进行数据交换的方式。一方面为局域网内开发板与 PC 机或其他网络终端进行数据交换、资源共享。另一方面为开发板与广域网进行网络连接、数据传输。

15.1 广域网连接准备-修改开发板 IP 地址

本开发平台默认的 IP：192.168.1.1；DHCP 服务器是开启的，如果要修改开发平台的 IP 地址，操作如下：

（1）系统开启后，在超级终端键入“ifconfig”，回车，出现如图 4.1- 1 信息，可以看到本开发板的 IP 为 192.168.1.1，

```
root@MicroCloud:/#
root@MicroCloud:/# ifconfig
br-lan    Link encap:Ethernet  HWaddr 00:9A:D5:53:C3:00
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fd92:a3ee:b0b2::1/60 Scope:Global
          inet6 addr: fe80::29a:d5ff:fe53:c300/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:551 errors:0 dropped:0 overruns:0 frame:0
          TX packets:202 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:47503 (46.3 KiB)  TX bytes:31529 (30.7 KiB)
```

图 4.1- 1IP 等信息

（2）在超级终端键入“uci set network.lan.ipaddr=192.168.Y.x”在子网掩码为 255.255.255.0 时，Y 不同代表网段不同，x 代表主机号。例如我要修改开发平台的 IP 地址为 192.168.2.1，则键入“uci set network.lan.ipaddr=192.168.2.1”回车，再键入“uci commit network”回车，最后键入“reboot”重启开发平台，如图 4.1- 2 所示。（注：uci 为设置有线或无线连接的相关选项的命令）

```
root@MicroCloud:/#
root@MicroCloud:/# uci set network.lan.ipaddr=192.168.2.1
root@MicroCloud:/# uci commit network
root@MicroCloud:/# reboot
```

图 4.1- 2 修改 IP

（3）待系统重启后，键入“ifconfig”确定修改是否成功，如图 4.1- 3 所示。

```
root@MicroCloud:/#
root@MicroCloud:/# ifconfig
br-lan    Link encap:Ethernet  HWaddr 00:9A:D5:53:C3:00
          inet addr:192.168.2.1  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fd92:a3ee:b0b2::1/60 Scope:Global
          inet6 addr: fe80::29a:d5ff:fe53:c300/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:733 errors:0 dropped:0 overruns:0 frame:0
          TX packets:397 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:70011 (68.3 KiB)  TX bytes:104631 (102.1 KiB)
```

图 4.1- 3 修改成功

这时表明修改成功。

15.2 广域网连接准备-局域网内网络通信

局域网内进行网络通信时，PC 机或网络终端可以通过有线和无线 AP 接入点两种方式进行网络连接：（1）进行有线连接时，用一根网线将 PC 机的网口与开发板的 LAN 口连接起来即可；（2）进行无线连接时，PC 机或网络终端可以搜索 SSID 为 MicroCloud 的无线信号进行连接（开发板的 WiFi 默认是 AP 接入点模式），默认没有加密密码。

开发板 LAN 口 IP 默认为 192.168.1.1，DHCP 服务器默认是开启的。进行网络连接时，PC 机、网络终端可以配置为自动获取 IP 或者把他们的 IP 地址修改为 192.168.1.x（x 为任意设置，但不能为 0、1 和 255，当多个网络终端连入时，不能出现相同的 IP 地址），子网掩码为（255.255.255.0），这时使开发板与 PC 机、网络终端在同一网段，网络连接建立，才可以进行数据交换。

注意：如果修改了开发平台的 IP 地址，要确保开发板与 PC 机、网络终端在同一网段，才可以进行数据交换。比如修改开发平台 IP 为 192.168.2.1，PC 机或网络终端要配置为自动获取 IP 或者把 IP 地址修改为 192.168.2.x（x 为任意设置，但不能为 0、1 和 255）。具体操作如下：

有线连接（PC 机为 WIN7 系统）：

（1） 点击电脑“开始”——“控制面板”——“网络和 Internet”——“网络和共享中心”——“更改适配器设置”；

（2） 然后双击“本地连接”，在出现的对话框中双击“Internet 协议版本 4（TCP/IPv4）”选项卡；

（3） 在弹出的对话框，选中“自动获得 IP 地址”和“自动获得 DNS 服务器地址”。或者，选择“使用下面的 IP 地址”，然后在“IP 地址”栏写入：192.168.1.x，在“子网掩码”栏写入 255.255.255.0。选项栏“使用下面的 DNS 服务器地址”处不用填写。最后，点击确定。

图 4.2- 1 自动获取 DNS 服务器

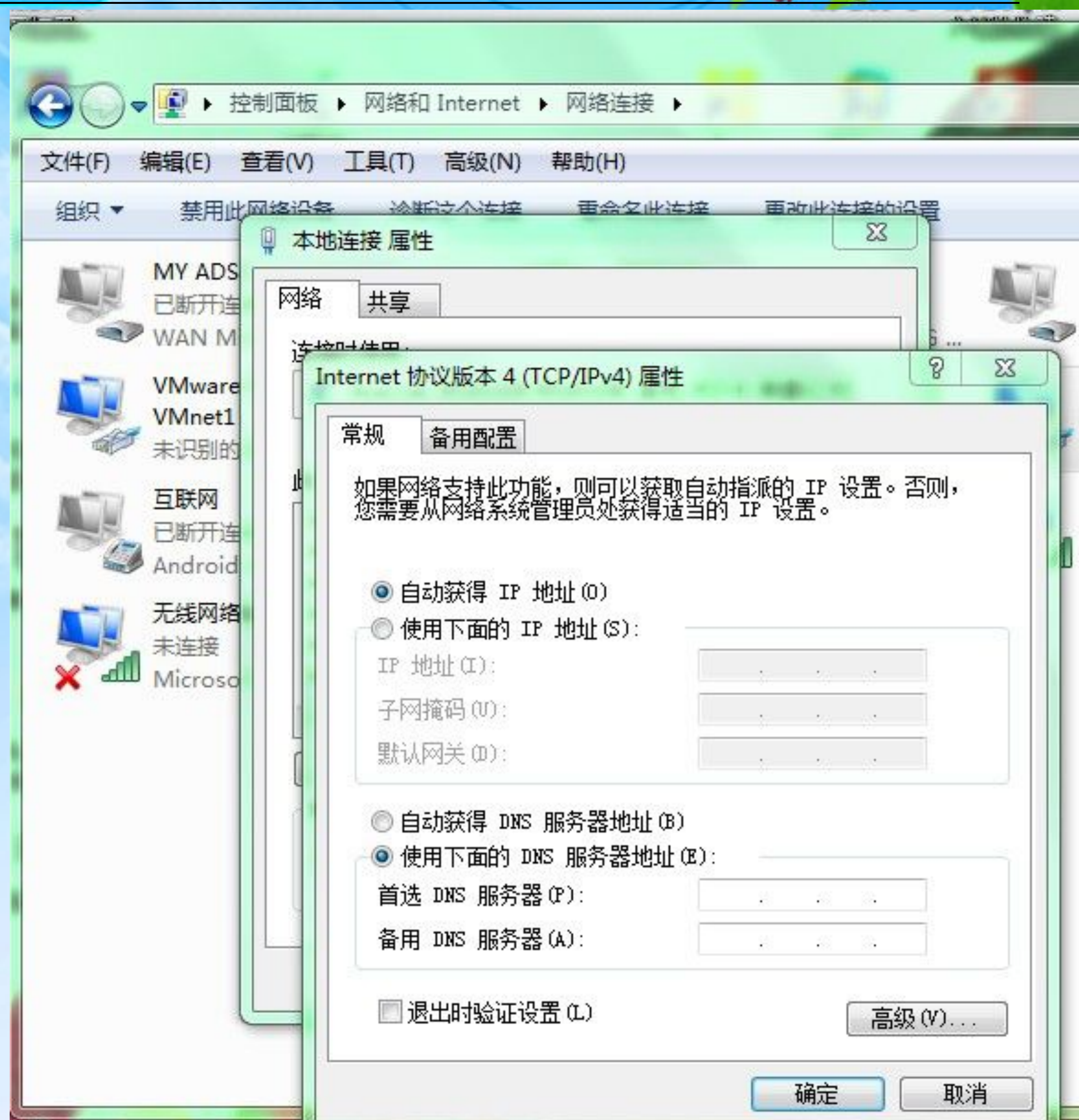


图 4.2- 1 自动获取 IP

手动写入 IP 地址如图 4.2- 2：

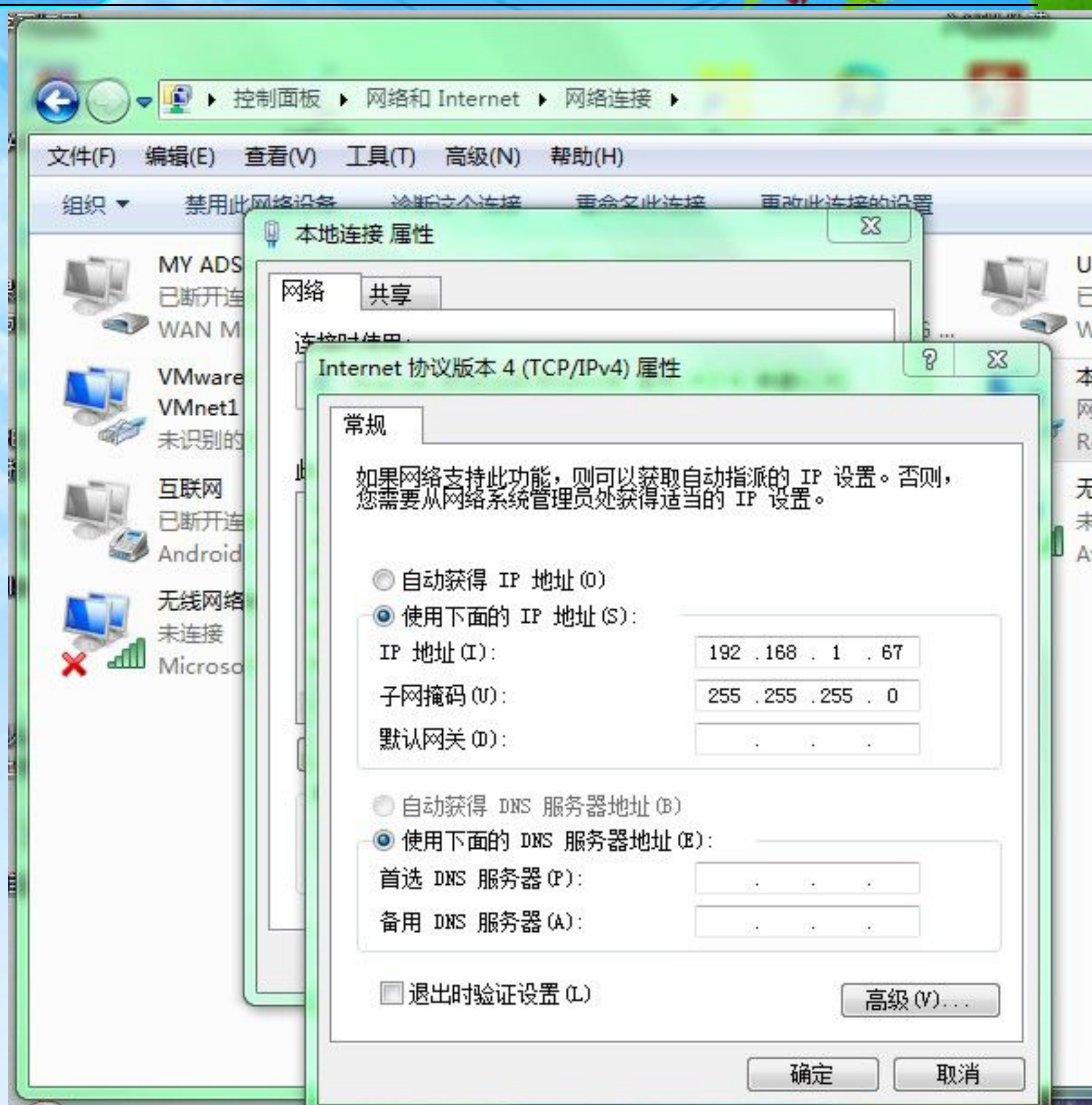


图 4.2- 2 手动配置 IP

无线连接：

（1） 点击电脑“开始”——“控制面板”——“网络和 Internet”——“网络和共享中心”——“更改适配器设置”；

（2） 然后右键点击“无线网络连接”，然后点击“属性”在出现的对话框中双击“Internet 协议版本 4 （TCP/IPv4）”选项卡

（3） 在弹出的对话框，选中“自动获得 IP 地址”和“自动获得 DNS 服务器地址”或者选择“使用下面的 IP 地址”，然后在“IP 地址”栏写入：192.168.1.x。“使用下面的 DNS 服务器地址”处不用填写，然后点击确定，搜索 SSID 为 MicroCloud 的无线信号进行连接。注意事项与有线连接时相同。



15.3 开发板广域网连接模式

15.3.1 有线广域网连接模式

有线上网方式是指通过开发板的 WAN 口来上网，网线的一端接开发板的 WAN 口，而网线的另一端接家庭路由器 LAN 口出来的网线。下面就这种情况进行介绍以及相关配置说明（前提是家庭已经可以通过有线上网，而且有路由器装置，路由器开启 DHCP 服务器）。

首先将网线的一端插入开发板的 WAN 口，另一端插入家庭路由器的一个 LAN 口，此时家庭路由器开启 DHCP 服务器，将开发板的 WAN 口默认配置成 DHCP 客户端模式，同时注意，如果家庭路由器 LAN 口 IP 地址已经是 192.168.1.x（x 可能为除 0、255 的任意值），子网掩码为 255.255.255.0，则应把开发板的 LAN 口 IP 地址修改成不同网段的 IP，修改开发板 IP 为 192.168.Y.x（Y 值不为 1，x 为除 0、255 的任意值，参考本章 3.1 节）。因为当子网掩码同时为 255.255.255.0 时，IP 地址 192.168.Y.x 中的 Y 值不同，表示开发板的 WAN 口和 LAN 口网段不同。才能使开发板上网。

15.3.2 无线广域网连接模式

无线上网方式为通过 USB 3G 网卡来进行上网。首先不要插入 3G 网卡，将开发板上电，等系统运行起来后，通过超级终端查看开发板文件系统/dev 目录下的内容，操作步骤如下:

（1）在超级终端中键入“ls /dev”，然后回车，如图 4.3- 1 所示：

```
root@MicroCloud:/dev# ls
audio               mtd0          mtdblock1           sda1
autofs              mtd0ro        mtdblock2           sda2
bus                 mtd1          mtdblock3           sg0
console             mtd1ro        mtdblock4           shm
cpu_dma_latency     mtd2          mtdblock5           snd
dsp                 mtd2ro        network_latency     tty
etherd              mtd3          network_throughput  ttyS0
full                mtd3ro        null                ttyS1
input               mtd4          ppp                 urandom
kmsg                mtd4ro        ptmx                video0
log                 mtd5          pts                 watchdog
mem                 mtd5ro        random              watchdog0
mixer               mtdblock0     sda                 zero
```

图 4.3- 1 查看设备

这时，你会看到/dev 文件夹下的设备，其中 ttyS0 和 ttyS1 是两个串口设备；

（2）然后在开发板上插入 USB 3G 网卡，在超级终端中键入“usbmode -s 12d1:1505”（usbmode 为识别 USB 设备命令），然后回车，如图 4.3- 2 所示：



```
root@MicroCloud:/# usbmode -s 12d1:1505
[  686.310000] usb 1-1.4.2: USB disconnect, device number 10
glob failed on /etc/init.d/usbmode
root@MicroCloud:/# glob failed on /etc/init.d/usbmode
[  688.000000] usb 1-1.4.2: new full-speed USB device number 11 using ehci-platform
[  688.140000] usb-storage 1-1.4.2:1.0: USB Mass Storage device detected
[  688.170000] option 1-1.4.2:1.0: GSM modem (1-port) converter detected
[  688.210000] usb 1-1.4.2: GSM modem (1-port) converter now attached to ttyUSB2
[  688.250000] usb-storage 1-1.4.2:1.1: USB Mass Storage device detected
[  688.270000] usb-storage 1-1.4.2:1.2: USB Mass Storage device detected
[  688.290000] option 1-1.4.2:1.2: GSM modem (1-port) converter detected
[  688.300000] usb 1-1.4.2: GSM modem (1-port) converter now attached to ttyUSB3
[  688.340000] usb-storage 1-1.4.2:1.3: USB Mass Storage device detected
[  688.360000] option 1-1.4.2:1.3: GSM modem (1-port) converter detected
[  688.360000] usb 1-1.4.2: GSM modem (1-port) converter now attached to ttyUSB4
[  688.420000] usb-storage 1-1.4.2:1.4: USB Mass Storage device detected
[  688.460000] scsi13 : usb-storage 1-1.4.2:1.4
glob failed on /etc/init.d/usbmode
glob failed on /etc/init.d/usbmode
glob failed on /etc/init.d/usbmode
glob failed on /etc/init.d/usbmode
[  689.470000] scsi 13:0:0:0: Direct-Access     OEM      Qualcom EVDOReVA 2.31 PQ: 0 ANSI: 0
[  689.520000] sd 13:0:0:0: Attached scsi generic sg2 type 0
[  689.550000] sd 13:0:0:0: [sdc] Attached SCSI removable disk
glob failed on /etc/init.d/usbmode
glob failed on /etc/init.d/usbmode
```

图 4.3- 2 网卡插入的信息

这时，超级终端会打印如上图信息，出现 ttyUSB2、ttyUSB3 和 ttyUSB4，（如果没有出现 ttyUSB2、ttyUSB3 和 ttyUSB4 等信息，请检查 USB 3G 网卡是否插好，然后多键入几次“usbmode -s 12d1:1505”命令，直到出现 tyUSB2、ttyUSB3 和 ttyUSB4 等信息（注：这里的 ttyUSBx 中的 x 是不固定的，和你插入的 USB 设备的数量有关，如果你之前插入了摄像头等，x 的数目就会顺延）；

（3）超级终端中键入“uci set network.MicroCloud_3g.device=/dev/ttyUSB2”，回车，再键入“uci commit network”命令，回车，如图 4.3- 3 所示：

```
root@MicroCloud:/#
root@MicroCloud:/# uci set network.MicroCloud_3g.device=/dev/ttyUSB2
root@MicroCloud:/# uci commit network
root@MicroCloud:/#
```

图 4.3- 3 配置网卡接口

（4）在超级终端中键入“/etc/init.d/network restart”，然后回车，如图 4.3- 4 启动网络配置所示：

```
root@MicroCloud:/# /etc/init.d/network restart
[ 1420.350000] br-lan: port 2(wlan0) entered disabled state
[ 1420.370000] br-lan: port 1(eth0.1) entered disabled state
[ 1420.390000] device eth0.1 left promiscuous mode
[ 1420.390000] br-lan: port 1(eth0.1) entered disabled state
[ 1420.410000] device wlan0 left promiscuous mode
[ 1420.410000] br-lan: port 2(wlan0) entered disabled state
[ 1421.060000] device eth0 left promiscuous mode
root@MicroCloud:/# [ 1423.030000] rt305x-esw 10110000.esw: link changed 0x00
[ 1424.850000] device eth0.1 entered promiscuous mode
[ 1424.860000] device eth0 entered promiscuous mode
[ 1424.870000] IPv6: ADDRCONF(NETDEV_CHANGE): eth0.1: link becomes ready
[ 1424.890000] rt305x-esw 10110000.esw: link changed 0x10
[ 1424.900000] br-lan: port 1(eth0.1) entered forwarding state
[ 1424.920000] br-lan: port 1(eth0.1) entered forwarding state
[ 1426.530000] rt305x-esw 10110000.esw: link changed 0x10
[ 1426.920000] br-lan: port 1(eth0.1) entered forwarding state
[ 1431.390000] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
[ 1431.410000] device wlan0 entered promiscuous mode
[ 1431.470000] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
[ 1431.860000] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
[ 1431.870000] br-lan: port 2(wlan0) entered forwarding state
[ 1431.890000] br-lan: port 2(wlan0) entered forwarding state
[ 1433.890000] br-lan: port 2(wlan0) entered forwarding state
```

图 4.3- 4 启动网络配置

超级终端出现如上图信息，代表网络正在重启，等待一端时间，USB3G 网卡指示灯快速闪烁，表明 USB 3G 网卡正在拨号，然后在超级终端中键入“ping

www.baidu.com”，如图 4.3- 5 所示：

```
root@MicroCloud:/#
root@MicroCloud:/#
root@MicroCloud:/# ping www.baidu.com
PING www.baidu.com (220.181.112.143): 56 data bytes
64 bytes from 220.181.112.143: seq=0 ttl=56 time=766.831 ms
64 bytes from 220.181.112.143: seq=1 ttl=56 time=182.416 ms
64 bytes from 220.181.112.143: seq=2 ttl=56 time=179.875 ms
64 bytes from 220.181.112.143: seq=3 ttl=56 time=269.032 ms
64 bytes from 220.181.112.143: seq=4 ttl=56 time=177.955 ms
64 bytes from 220.181.112.143: seq=5 ttl=56 time=335.705 ms
64 bytes from 220.181.112.143: seq=6 ttl=56 time=172.993 ms
64 bytes from 220.181.112.143: seq=7 ttl=56 time=196.197 ms
^C
--- www.baidu.com ping statistics ---
9 packets transmitted, 8 packets received, 11% packet loss
round-trip min/avg/max = 172.993/285.125/766.831 ms
root@MicroCloud:/#
```

图 4.3- 5 ping 百度

在超级终端中出现如上图内容代表网络已经建立，开发板这时可以上网。（键入“ping www.baidu.com”时，如果网络建立，超级终端会不停地显示 ttl 信息，这时在终端中键入“ctrl+c”命令。

注意：如果在插入 USB 3G 网卡之前，“ls /dev”时出现了 ttyUSB0、ttyS0、ttyUSB2……等多个 ttyUSB 设备，这时插入 USB 3G 网卡执行步骤（2）时，又会出现 3 个 ttyUSBx 设备，这时按如下操作：

假如没有插入 USB 3G 网卡之前，/dev 下有 ttyUSB0、ttyUSB1、ttyUSB2 三个设备，这时执行步骤（2），超级终端会出现 ttyUSB3、ttyUSB4、ttyUSB5 三个新的 ttyUSB）。此刻再执行步骤（3）时，应键入“uci set network.MicroCloud_3g.device=/dev/ttyUSBx”,（ttyUSBx 中的 x 代表插入 USB 3G 网卡后执行步骤（2）新出现三个 ttyUSBx 中的第一个 ttyUSB 设备号），再键入“uci commit network”，然后再执行步骤（4），即可上网。操作如图 4.3- 6 新添一个 USB 设备：

键入“ls /dev”命令，回车：

```
root@MicroCloud:/# cd /dev
root@MicroCloud:/dev# ls
autofs              mtd2                network_latency     snd
bus                 mtd2ro              network_throughput  tty
console             mtd3                null                ttyS0
cpu_dma_latency     mtd3ro              ppp                 ttyS1
etherd              mtd4                ptmx                ttyUSB0
full                mtd4ro              pts                 ttyUSB1
input               mtd5                random              ttyUSB2
kmsg                mtd5ro              sda                 urandom
log                 mtdblock0           sda1                watchdog
mem                 mtdblock1           sda2                watchdog0
mtd0                mtdblock2           sdb                 zero
mtd0ro              mtdblock3           sg0
mtd1                mtdblock4           sg1
mtd1ro              mtdblock5           shm
```

图 4.3- 6 新添一个 USB 设备

插入 USB 3G 网卡，键入“usbmode -s 12d1:1505”命令，回车，如图 4.3- 7 所示：

```
root@MicroCloud:/dev#
root@MicroCloud:/dev# usbmode -s 12d1:1505
root@MicroCloud:/dev# [ 4488.010000] usb 1-1.4.1: USB disconnect, device number 13
glob failed on /etc/init.d/usbmode
glob failed on /etc/init.d/usbmode
[ 4489.470000] usb 1-1.4.1: new full-speed USB device number 14 using ehci-platform
[ 4489.610000] usb-storage 1-1.4.1:1.0: USB Mass Storage device detected
[ 4489.640000] option 1-1.4.1:1.0: GSM modem (1-port) converter detected
[ 4489.680000] usb 1-1.4.1: GSM modem (1-port) converter now attached to ttyUSB3
[ 4489.680000] usb-storage 1-1.4.1:1.1: USB Mass Storage device detected
[ 4489.740000] usb-storage 1-1.4.1:1.2: USB Mass Storage device detected
[ 4489.780000] option 1-1.4.1:1.2: GSM modem (1-port) converter detected
[ 4489.800000] usb 1-1.4.1: GSM modem (1-port) converter now attached to ttyUSB4
[ 4489.820000] usb-storage 1-1.4.1:1.3: USB Mass Storage device detected
[ 4489.840000] option 1-1.4.1:1.3: GSM modem (1-port) converter detected
[ 4489.860000] usb 1-1.4.1: GSM modem (1-port) converter now attached to ttyUSB5
[ 4489.870000] usb-storage 1-1.4.1:1.4: USB Mass Storage device detected
[ 4489.910000] scsi19 : usb-storage 1-1.4.1:1.4
glob failed on /etc/init.d/usbmode
```

图 4.3- 7 网卡插入信息

出现了 ttyUSB3、ttyUSB4、ttyUSB5，然后键入“uci set network.MicroCloud_3g.device=/dev/ttyUSB3”命令，回车；键入“uci commit network”命令，回车，如图 4.3- 8 所示：

```
root@MicroCloud:/# uci set network.MicroCloud_3g.device=/dev/ttyUSB3
root@MicroCloud:/# uci commit network
root@MicroCloud:/#
```

图 4.3- 8 配置网卡接口

键入“/etc/init.d/network restart”命令，回车：

```
root@MicroCloud:/#
root@MicroCloud:/# /etc/init.d/network restart
[ 1015.710000] br-lan: port 2(wlan0) entered disabled state
[ 1015.730000] br-lan: port 1(eth0.1) entered disabled state
[ 1015.760000] device eth0.1 left promiscuous mode
[ 1015.770000] br-lan: port 1(eth0.1) entered disabled state
[ 1015.810000] device wlan0 left promiscuous mode
[ 1015.810000] br-lan: port 2(wlan0) entered disabled state
[ 1016.490000] device eth0 left promiscuous mode
root@MicroCloud:/# [ 1018.290000] rt305x-esw 10110000.esw: link changed 0x00
[ 1019.850000] device eth0.1 entered promiscuous mode
[ 1019.850000] device eth0 entered promiscuous mode
[ 1019.900000] br-lan: port 1(eth0.1) entered forwarding state
[ 1019.910000] br-lan: port 1(eth0.1) entered forwarding state
[ 1020.150000] rt305x-esw 10110000.esw: link changed 0x10
[ 1020.250000] rt305x-esw 10110000.esw: link changed 0x12
[ 1020.900000] IPv6: ADDRCONF(NETDEV_CHANGE): eth0.1: link becomes ready
[ 1021.890000] rt305x-esw 10110000.esw: link changed 0x12
[ 1022.210000] br-lan: port 1(eth0.1) entered forwarding state
[ 1023.200000] rt305x-esw 10110000.esw: link changed 0x12
[ 1025.150000] rt305x-esw 10110000.esw: link changed 0x12
[ 1025.720000] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
[ 1025.740000] device wlan0 entered promiscuous mode
```

图 4.3- 9 启动网络

操作完毕。

第 16 章 lftp 网络文件传输协议

lftp 是一个功能强大的下载工具，它支持访问文件的协议：ftp，ftps，http，https，hftp，fish.（其中 ftps 和 https 需要在编译的时候包含 openssl 库）。我们的开发板便支持 lftp 上传和下载文件的功能。

16.1 lftp 命令介绍

首先是登陆服务器的命令，有以下几种：

- ❖ lftp ftp://user:password@site:port
- ❖ lftp user:password@site:port
- ❖ lftp site -p port -u user,password
- ❖ lftp site:port -u user,password

下面我们看一下 lftp 常用的命令：

- ❖ ls 显示远端文件列表(!ls 显示本地文件列表)。
- ❖ cd 切换远端目录(LCD 切换本地目录)。
- ❖ get 下载远端文件。
- ❖ mget 下载远端文件(可以用通配符也就是 *)。
- ❖ pget 使用多个线程来下载远端文件, 预设为五个。
- ❖ mirror 下载/上传(mirror -R)/同步 整个目录。
- ❖ put 上传文件。
- ❖ mput 上传多个文件(支持通配符)。
- ❖ mv 移动远端文件(远端文件改名)。
- ❖ rm 删除远端文件。
- ❖ mrm 删除多个远端文件(支持通配符)。
- ❖ mkdir 建立远端目录。
- ❖ rmdir 删除远端目录。
- ❖ pwd 显示目前远端所在目录(lpwd 显示本地目录)。
- ❖ du 计算远端目录的大小
- ❖ !执行本地 shell 的命令(由于 lftp 没有 lls, 故可用 !ls 来替代)
- ❖ LCD 切换本地目录
- ❖ lpwd 显示本地目录
- ❖ alias 定义别名
- ❖ bookmark 设定书签。
- ❖ exit 退出 lftp

16.2 lftp 命令的举例

连接好串口线，打开 SecureCRT，给开发板上电，配置好网络（有线连接，无线连接均可，只要保证上位机和开发板网络畅通即可，可参看上一章内容）。

上位机打开 FileZilla server，如图 5.2- 1 所示：



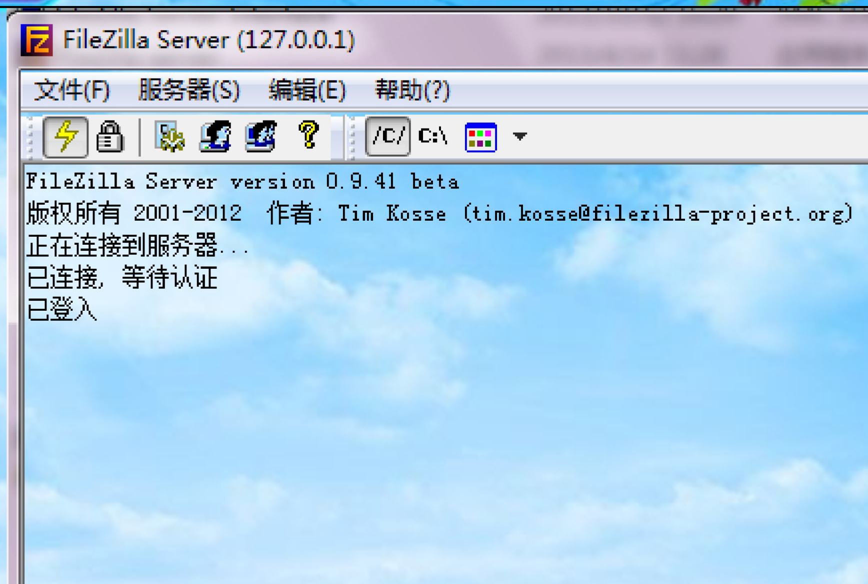


图 5.2- 1 登陆界面

开发板启动完毕之后，在 SecureCRT 里输入命令：lftp 用户名：密码@IP，我们这里以用户名 gxt，密码为 1，IP 为 10.3.9.198 为例，如图 5.2- 2 所示：

```
root@MicroCloud:/# lftp gxt:1@10.3.9.198
lftp gxt@10.3.9.198:~>
```

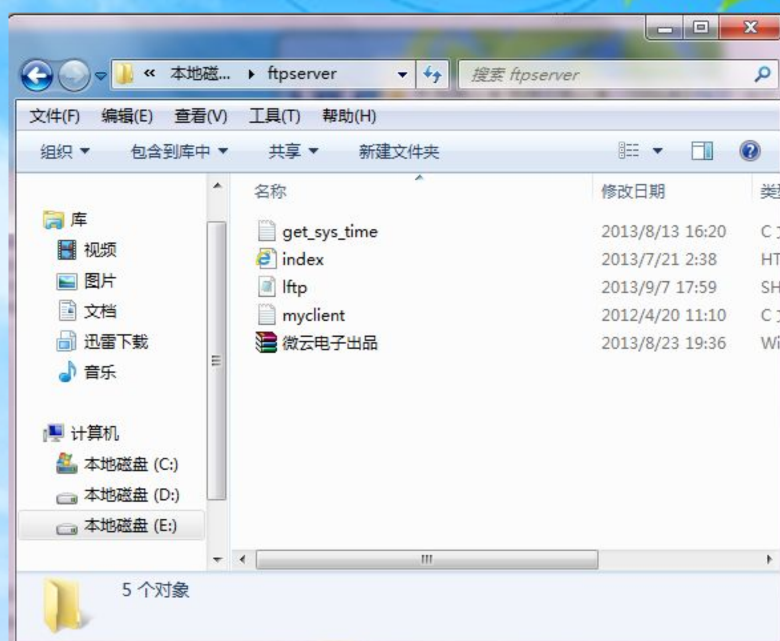
图 5.2- 2 建立连接

当出现上图所示时，说明 ftp 连接建立成功，ls 命令可以查看上位机共享文件夹内的文件，如图 5.2- 3 所示：

```
root@MicroCloud:/# lftp gxt:1@10.3.9.198
lftp gxt@10.3.9.198:~> ls
[ 2946.870000] nf_conntrack: automatic helper assignment is deprecated and
-rw-r--r-- 1 ftp ftp           231 Aug 13  2013 get_sys_time.c
-rw-r--r-- 1 ftp ftp           419 Jul 21  2013 index.html
-rw-r--r-- 1 ftp ftp            98 Sep 07  2013 lftp.sh
-rw-r--r-- 1 ftp ftp          1115 Apr 20  2012 myclient.c
-rw-r--r-- 1 ftp ftp     478738294 Aug 23  2013 褰 稹鐢靛瑧鍑哄揣.rar
lftp gxt@10.3.9.198:/>
```

图 5.2- 3 显示共享文件夹内容

和我们设置的 E:\ftpserver 文件夹下内容是一样的：



我们用命令从共享文件夹下载一个文件试试：

```
lftp gxt@10.3.9.198:/> get index.html
419 bytes transferred in 4 seconds (98b/s)
lftp gxt@10.3.9.198:/>
```

图 5.2- 4 下载文件

输入退出命令 quit，回到原来的环境

```
lftp gxt@10.3.9.198:/> quit
root@MicroCloud:/#
```

图 5.2- 5 退出

再用 ls 查看当前的文件夹：

```
root@MicroCloud:/# ls
Program      etc          lost+found   proc         sbin         usr
bin          index.html   mnt          rom          sys          var
dev          lib          overlay      root         tmp          www
root@MicroCloud:/#
```

图 5.2- 6 显示文件

可以看到目录下多了一个 index.html 文件。

这里我们是从根目录进行的 lftp 连接，所以在 lftp 连接之后，需要上传(put 或 mput)的文件必须是根目录下的，否则会提示无法找到文件，同样从共享文件夹下载(get 或 mget)的文件也会相应的放在根目录下。如果我们进入到 www 目录下来进行 lftp 连接，那么上传下载的操作就会在 www 目录下进行，这是需要我们注意的。

第 17 章 网络摄像之 Web 实时监控

Mjpg-streamer，是用于从 webcam 摄像头采集图像的软件，把他们以流的形式通过基于 IP 的网络传输到浏览器，如 Firehox（火狐浏览器），Chrome（谷歌浏览器），Cambozola，VLC 播放器，Windows 的移动设备或者其他拥有浏览器的移动设备。我们开发板的系统支持这一款软件。

17.1 Mjpg-streamer 命令介绍

图 6.1- 1 是查看系统帮助的介绍

```
root@MicroCloud:/dev# mjpg_streamer --help
-----------------------------------------------------------------------
Usage: mjpg_streamer
  -i | --input "<input-plugin.so> [parameters]"
  -o | --output "<output-plugin.so> [parameters]"
 [-h | --help ]........: display this help
 [-v | --version ].....: display version information
 [-b | --background]...: fork to the background, daemon mode
-----------------------------------------------------------------------
Example #1:
 To open an UVC webcam "/dev/video1" and stream it via HTTP:
  mjpg_streamer -i "input_uvc.so -d /dev/video1" -o "output_http.so"
-----------------------------------------------------------------------
Example #2:
 To open an UVC webcam and stream via HTTP port 8090:
  mjpg_streamer -i "input_uvc.so" -o "output_http.so -p 8090"
-----------------------------------------------------------------------
Example #3:
 To get help for a certain input plugin:
  mjpg_streamer -i "input_uvc.so --help"
-----------------------------------------------------------------------
In case the modules (=plugins) can not be found:
 * Set the default search path for the modules with:
   export LD_LIBRARY_PATH=/path/to/plugins,
 * or put the plugins into the "/lib/" or "/usr/lib" folder,
 * or instead of just providing the plugin file name, use a complete
   path and filename:
   mjpg_streamer -i "/path/to/modules/input_uvc.so"
-----------------------------------------------------------------------
```

图 6.1- 1Mjpg-streamer 系统介绍

mjpg_streamer -i "input_uvc.so -r 352x288 -f 15 -q 80 -y"   -o "output_http.so -p 8080 -w /www"这个命令是比较完整的启动摄像头的命令，表示让 mjpeg-streamer 以默认的 352x288 分辨率、15fps 显示图像，并且监听 8080 端口的 http 请求。其自带的一个小型 web 页面所在的地址为/www。

17.2 Mjpg-streamer 命令的举例

同样上位机和开发板保证正常的网络通信，将摄像头插入 HUB 中的一个 USB 插孔，如图 6.2- 1 所示：



图 6.2- 1 摄像头实物

可以看到在 SecureCRT 终端里出现如图 6.2- 2 信息：

```
root@MicroCloud:/# [  262.190000] usb 1-1.4.2: new high-speed USB device number 10 using ehci-platform
[  262.350000] usb 1-1.4.2: no of_node; not parsing pinctrl DT
[  262.360000] uvcvideo 1-1.4.2:1.0: no of_node; not parsing pinctrl DT
[  262.380000] uvcvideo: Found UVC 1.00 device USB 2.0 Camera (0c45:6340)
[  262.420000] input: USB 2.0 Camera as /devices/101c0000.ehci/usb1/1-1/1-1.4/1-1.4.2/1-1.4.2:1.0/input/input1
[  262.480000] snd-usb-audio 1-1.4.2:1.2: no of_node; not parsing pinctrl DT
[  262.510000] 10:3:1: cannot get freq at ep 0x84
```

图 6.2- 2 插入摄像头显示的信息

查看/dev 里的设备符，会出现 video0 的设备，就是我们的摄像头：

```
cd /dev
root@MicroCloud:/dev# ls
audio               mtd0ro         mtdblock3           shm
autofs              mtd1           mtdblock4           snd
bus                 mtd1ro         mtdblock5           tty
console             mtd2           network_latency     ttyS0
cpu_dma_latency     mtd2ro         network_throughput  ttyS1
dsp                 mtd3           null                ttyUSB0
etherd              mtd3ro         ppp                 ttyUSB1
full                mtd4           ptmx                urandom
input               mtd4ro         pts                 video0
kmsg                mtd5           random              watchdog
log                 mtd5ro         sda                 watchdog0
mem                 mtdblock0      sdb                 zero
mixer               mtdblock1      sg0
mtd0                mtdblock2      sg1
```

图 6.2- 3 设备信息

输入命令 mjpg_streamer -i "input_uvc.so -r 352x288 -f 15 -q 80 -y" -o "output_http.so -p 8080 -w /www"启动摄像头：

```
root@MicroCloud:/dev# mjpg_streamer -i "input_uvc.so -r 352x288 -f 15 -q 80 -y"
 -o "output_http.so -p 8080 -w /www"
MJPG Streamer Version: svn rev: exported
 i: Using V4L2 device.: /dev/video0
 i: Desired Resolution: 352 x 288
 i: Frames Per Second.: 15
 i: Format............: YUV
 i: JPEG Quality......: 80
Adding control for Pan (relative)
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
Adding control for Tilt (relative)
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
Adding control for Pan Reset
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
Adding control for Tilt Reset
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
Adding control for Pan/tilt Reset
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
Adding control for Focus (absolute)
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
mapping control for Pan (relative)
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Tilt (relative)
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Pan Reset
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Tilt Reset
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Pan/tilt Reset
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Focus (absolute)
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for LED1 Mode
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for LED1 Frequency
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Disable video processing
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Raw bits per pixel
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
 o: www-folder-path...: /www/
 o: HTTP TCP port.....: 8080
 o: username:password.: disabled
 o: commands..........: enabled
[ 7461.140000] rt305x-esw 10110000.esw: link changed 0x02
```

图 6.2- 4 启动摄像头所显示的信息

在上位机的谷歌或者火狐浏览器地址栏输入：
http://192.168.1.1:8080/?action=stream
就可以看到摄像头的监控画面了，如图 6.2- 5 所示：

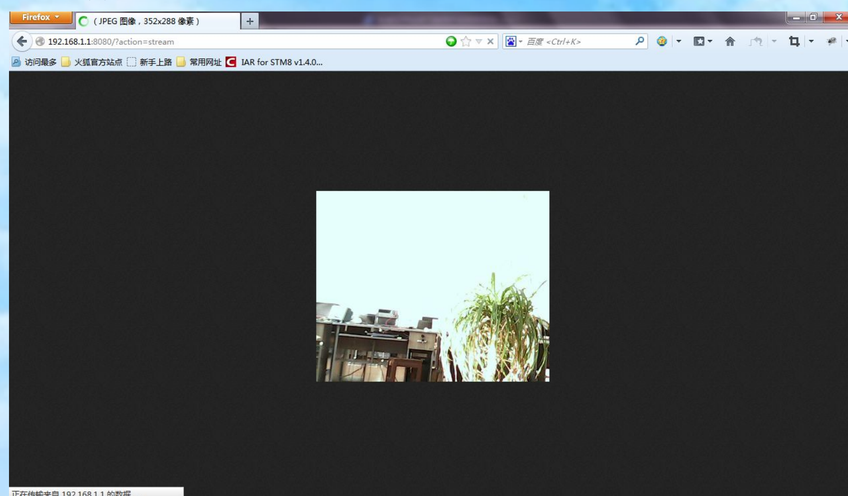


图 6.2- 5 浏览器的画面 1

修改 mjpg_streamer -i "input_uvc.so -r 352x288 -f 15 -q 80 -y" -o "output_http.so -p 8080 -w /www"各项参数，可以调节图像。比如，将分辨率改为 600*480，在终端输入命令 mjpg_streamer -i "input_uvc.so -r 600x480 -f 15 -q 80 -y" -o "output_http.so -p 8080 -w /www"，此时浏览器中图像显示如图 6.2- 6 所示：

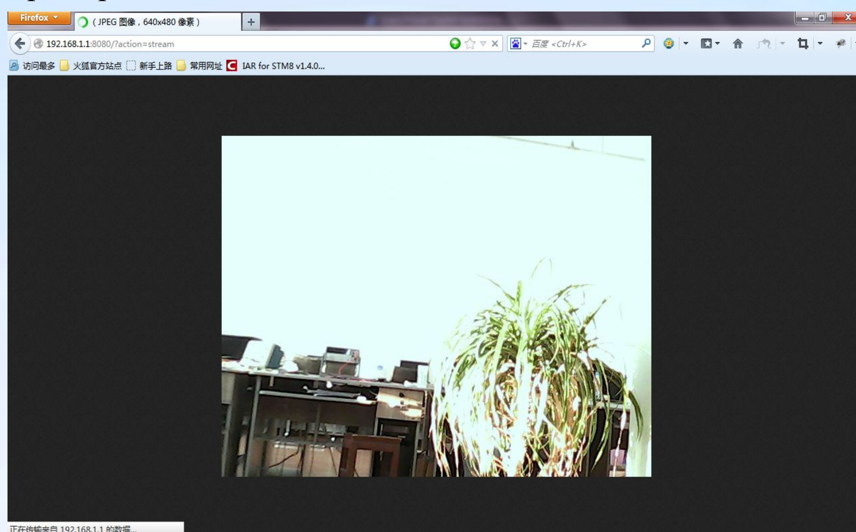


图 6.2- 6 浏览器的画面 2

明显可以看到图像变大了许多，这就是我们修改分辨率的结果。

第 18 章 微型 Web 服务器构建

通过上一章我们知道，可以通过浏览器查看摄像头的监视画面，那么是否可以查看其它的网页信息呢？当然可以，我们的开发板实际上是一个小型的 web 服务器，www 目录就是用来实现这一功能的，只要把编辑好的网页文件（html）存放到 www 目录下，然后在浏览器地址栏里输入 IP/网页文件名，就可以查看网

页了。比如，我们用 WinSCP 工具上传一个 picture.htm 的文件：

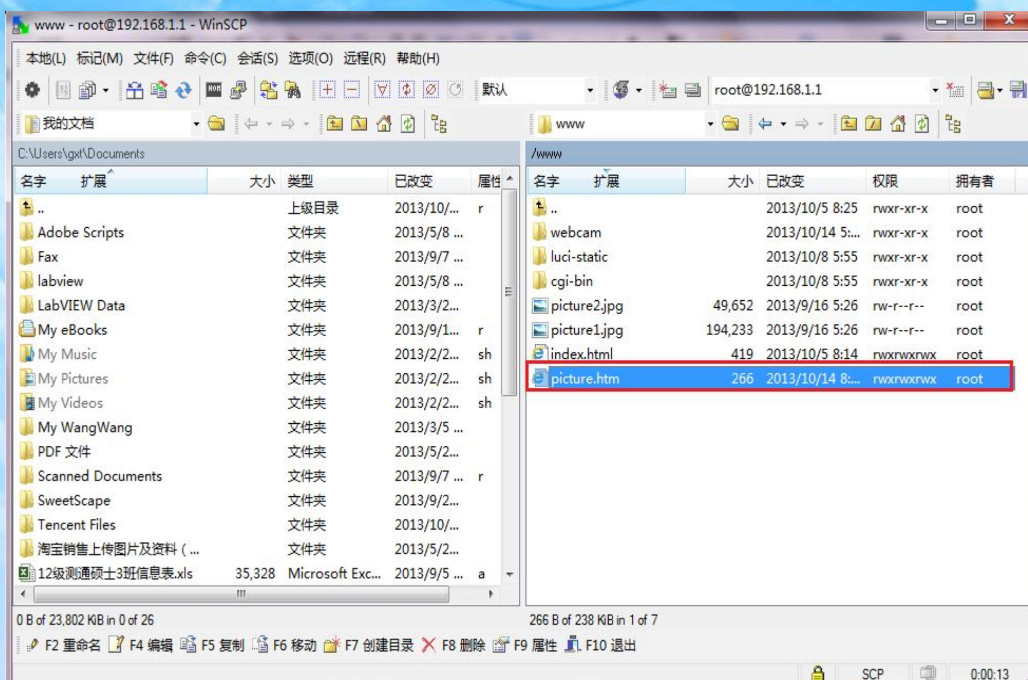


图 7.1- 1 上传的 picture.c 文件

然后在浏览器里输入 192.168.1.1/picture.htm 回车，可以看到网页信息了：



图 7.1- 2 网页信息

第 19 章 网络邮件及图片附件收发功能

19.1 网络邮件的发送

我们的开发板具有网络连接功能，通过配置发送邮件的软件，就可以发送邮



件。发送邮件的软件有很多种，我们这里是用 mutt 与 msmtp 两种软件配合来达到发送邮件的目的。

Mutt：mutt 是 linux 下的一个 email 软件。它跟一般的 windows 邮件软件不同，它不能直接发送邮件。Mutt 更像一个文件管理器，只不过它管理的是 email，把需要发送的邮件，按照需要的格式编辑起来。

Msmtp：msmtp 是一个开源的 smtp 服务器，smtp 即简单邮件传输协议，用来发送或中转发出的电子邮件。msmtp 在默认模式下，帮助 mutt 传送邮件。

19.1.1 发送邮件软件配置

因为邮箱账号因人而异，因此我们需要更改 mutt 和 msmtp 的配置文件，以达到我们的目的。

1、配置 mutt 的配置文件：

从 SecureCRT 命令行里进入到 etc 目录下

cd etc

更改 Muttrc 配置文件：命令行下输入 vi Muttrc,如图 8.1- 1 所示：

```
set sendmail="/usr/bin/msmtp"
set from="<            @139.com>"
set use_from=yes
set editor="vi"
set charset="gb2312"
~
~
~
~
```

图 8.1- 1 配置 Muttrc

set sendmail=”/usr/bin/msmtp”此语句的作用是将 mutt 所编好的邮件用 msmtp 来发送。

set from 指定自己的邮箱号，这里用的是 139 邮箱(也可用其他邮箱)。

set charset 指定邮件编码格式，“gb2312”代表发送显示汉字，若没有此语句，发送的汉字邮件在接收端会显示乱码。

2、配置 msmtp 的配置文件：

在安装 msmtp 软件时，会自动在 etc 目录下生成一个配置文件 msmtprc。

配置文件的内容如图 8.1- 2：

```
# Example for a system wide configuration file

# A system wide configuration file is optional.
# If it exists, it usually defines a default account.
# This allows msmtp to be used like /usr/sbin/sendmail.
account default

# The SMTP smarthost.
host smtp.139.com

# Construct envelope-from addresses of the form "user@oursite.example".
auto_from off
#maildomain oursite.example
from            @139.com
user
password

# Use TLS.
#tls on
auth login
#tls_trust_file /etc/ssl/certs/ca-certificates.crt

# Syslog logging with facility LOG_MAIL instead of the default LOG_USER.
logfile
syslog LOG_MAIL
~
~
```

图 8.1- 2 配置文件 msmtprc

在 from 后面黄色涂抹处填入自己的邮箱账号(这里用的是 139 邮箱)。user 指账号，即邮箱账号不加@139.com 部分。password 是邮箱密码。

19.1.2 发送邮件

配置完 mutt 和 msmtp 的配置文件之后，就可以发送邮件了。

在 SecureCRT 命令行下输入命令：

echo “123” | mutt -s “test” *********@139.com，这是我们发送邮件的命令。echo 是输出命令，“123”是发送邮件的正文(此处可以是自己想发送的任意内容)。mutt -s 用来指定发送邮件的主题，此处“test”就是我们本次发送邮件的主题。*********@139.com 是要发送邮件的目标邮箱，若有多个目标邮箱，则可以连续输入多个邮箱账号，各个邮箱账号之间用逗号隔开。

为了测试方便我们用同一个邮箱进行自发自收的测试，即发件人和收件人都是自己。在电脑上打开自己的 139 邮箱，就能在收件箱里看到刚刚发送的邮件了。如图 8.1- 3 所示：



图 8.1- 3 电脑收到的邮件

19.1.3 发送邮件之图片等附件

发送带有附件的邮件，只需在发送邮件命令上添加邮件附件属性即可。

例如，我们要把 xxx 目录下的 yyy 发送出去。发送命令为：

echo “邮件正文” | mutt -s “test” *********@139.com -a yyy < /xxx/yyy

-a 用于指定发送附件的属性，“yyy” 指定待发送附件的文件名，/xxx/yyy 指定附件所在的路径。

19.2 网络邮件的接收

开发板既可以发送邮件，也可以接收邮件。只需要配置好接收邮件的软件就可以了。接收邮件时我们用到的工具是 fetchmail 和 procmail。

Fetchmail：fetchmai 是一个下载邮件的应用软件，它支持 POP3、POP2 等大多数邮件协议，我们用它来接收邮件。

Procmail：procmail 用于过滤 email 邮件，它可以整理和处理大量的接收邮件，过滤掉垃圾邮件。Prcomail 一般和 fetchmail、mutt、sendmail 搭配使用。

19.2.1 接收邮件软件配置

1、配置 fetchmail 的配置文件 fetchmailrc

首先从 SecureCRT 命令行进入到 etc 目录：cd /etc
接着打开 fetchmailrc 文件：vi fetchmailrc，如图 8.2- 1 所示：

```
poll pop3.139.com protocol pop3 uidl username "          " password "        " to root keep
# smtphost localhost/110
mda "/usr/bin/procmail -d %T "



~
~
```

图 8.2- 1 配置文件 fetchmailrc

注意：因为这里用的是 139 邮箱，所以 pop3 后面是 139.com，如果要用其他邮箱时此处需要更改，例如用 163 邮箱，这里就要写成 pop3.163.com，pop3 代表邮件协议。

username 是自己的邮箱号，不加@139 后缀，password 是邮箱密码。这里的邮箱号和密码是收件人自己设置的邮箱号和密码，这里由于我们收发邮件用的是同一个邮箱，因此，同我们发送的邮箱号和密码是一致的。

Procmail 是一个邮箱过滤工具，不需要我们去配置配置文件。

2、其次，还需要我们创建一个 root 文件来存放邮件。因为 fetchmail 协议规定收到的邮件存放在/tmp/spool/mail 目录下的 root 文件里。但是开发板上并没有 spool/mail 目录和 root 文件，所以我们要自行创建。

在命令行下输入：

```
mkdir /tmp/spool                       //创建目录
mkdir /tmp/spool/mail
touch /tmp/spool/mail/root             //在/tmp/spool/mail 目录下新建 root 文件
chmod 777 /tmp/spool/mail/root         //修改 root 文件权限为可读可写可执行
```

第一行是创建 tmp 目录下的 spool 目录。

第二行是创建 spool 目录下的 mail 目录。

第三行是创建 mail 目录下的 root 文件。touch 是创建文件命令。

最后一行是给 root 文件赋予可执行权限。

通过创建 root 文件，我们就相当于给开发板做了个简易的邮箱，由于邮件接收。

19.2.2 接收邮件

配置好 fetchmail 的配置文件之后，我们就可以接收邮件了。

当完成 7.1 节中的邮件发送过程后(此时相当于自己给自己的邮箱发了一个邮件)，在 SecureCRT 命令行下输入接收邮件命令：

```
fetchmail -v -f /etc/fetchmailrc -P 110
```

fetchmail 是接收邮件命令，-v 是是显示输出调试信息，会把我们接收的信息显示出来,-f 是指定运行控制文件，就是按照我们的 fetchmailrc 文件里配置的去接收邮件,-P 是指定端号 110。

接收邮件命令执行之后，就可以查看邮件了，查看邮件命令：

```
cat /tmp/spool/mail/root
```

接收到的邮件如下图 8.2- 2 所示：

```
From root  Mon Sep  2 07:41:13 2013
X-Richmail-Antispam: sCL2rviOborhSzeOjpYxWEmrxqxSIjpywIkDhRS3bNKNy5SIjpgwYtQ=
X-RM-SPAM-FLAG:00000000
Received: from 121.195.184.79 [121.195.184.79]
        by 127.0.0.1 with POP3 (fetchmail-6.3.26)
        for <root@localhost> (single-drop); Mon, 02 Sep 2013 07:41:13 +0000 (GMT)
Received:from appfetion4 (unknown[172.16.72.141])
        by rmsmtp-cmudms-5-15-12020 (RichMail) with SMTP id 2ef45223f14ebb1-1bbb4;
        Mon, 02 Sep 2013 10:00:46 +0800 (CST)
X-RM-TRANSID:2ef45223f14ebb1-1bbb4
Date: Mon, 2 Sep 2013 10:00:46 +0800 (CST)
From: 15146843614@139.com
To: 15146843614@139.com
Message-ID: <1288814725.1409897137808724 6496.JavaMail.moSMS@appfetion4>
Subject: Re:test
MIME-Version: 1.0
Content-Type: text/plain;charset=GBK
Content-Transfer-Encoding: quoted-printable

1
```

图 8.2- 2 接收到的邮件

电脑发送的邮件如下图 8.2- 3 所示：

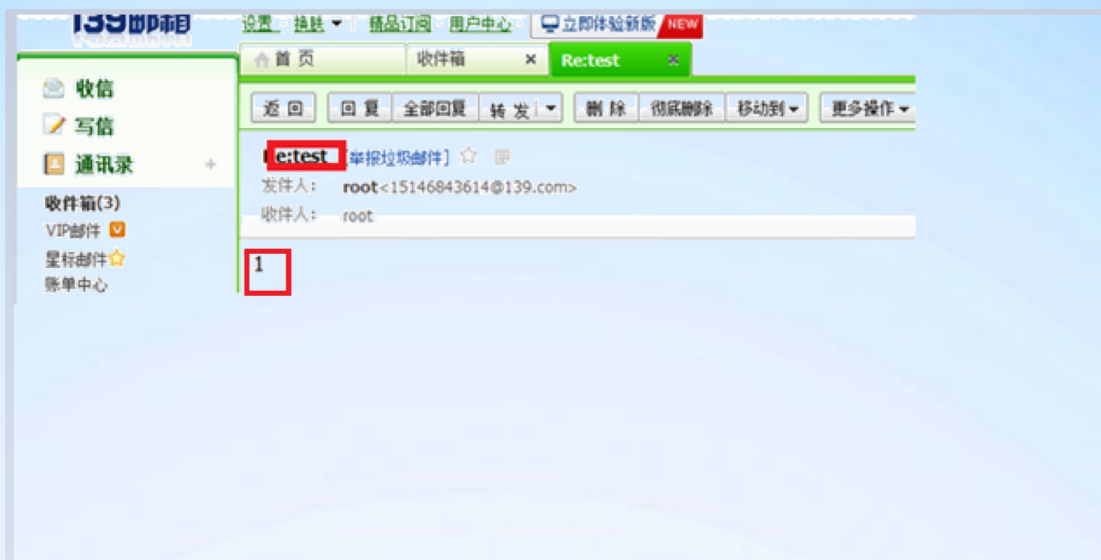


图 8.2- 3 电脑邮件

通过比较发送的邮件与接收到的邮件，我们会发现发送的邮件在 root 文件中出现了，因此，我们的邮件接收是成功的。

第 20 章 Samba 文件共享服务功能

20.1 Samba 共享介绍

SMB（Server Messages Block，信息服务块）是一种在局域网上共享文件和打印机的一种通信协议，它为局域网内的不同计算机之间提供文件及打印机等资

源的共享服务。这里配置 samba，可以让电脑和开发板之间共享文件。开发板里已经加入了 samba 共享功能。

20.2 查看 Samba 共享文件夹

启动我们的开发板之后，用 WinSCP 打开/etc/passwd 文件，如图 9.2- 1 所示：

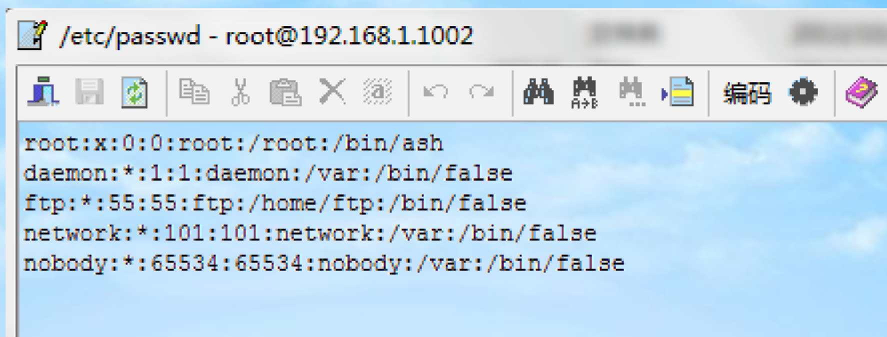


图 9.2- 1/etc/passwd

这里有一个用户叫 nobody 是 samba 共享文件登陆的用户名
如下命令来修改用户 nobody 的密码:
$ smbpasswd -a nobody

```
root@ 微云:/# smbpasswd -a nobody
New SMB password:
Retype SMB password:
root@ 微云:/# /etc/init.d/samba enable
root@ 微云:/# /etc/init.d/samba restart
```

图 9.2- 2 修改密码

打开我的电脑 在上方输入 \\192.168.1.1
注意：在输入 IP 之前一定不要把两个反斜杠丢掉\\，否则就会变成打开网页，而不是我们所要看到的共享内容，
输入用户名和密码（输入一次用户名和密码后，windows 便会记住密码，下次便不需要再输入用户名和密码了）。
可以看到共享的文件夹，如图 9.2- 3 所示：

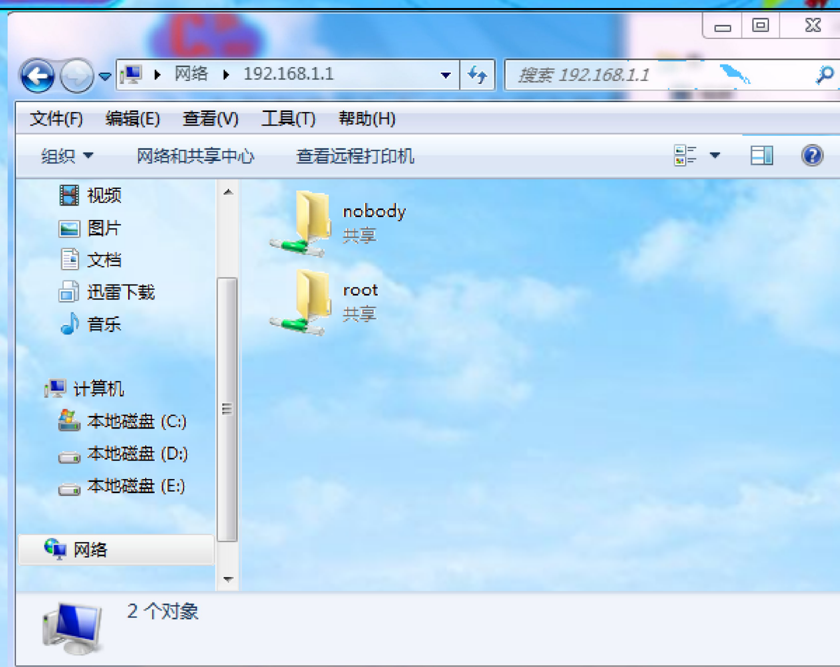


图 9.2- 3 共享文件夹内容

20.3 新增网络共享文件夹

想要添加自己想共享的文件夹，在配置文件/etc/config/samba 中加入图 9.3- 1 的内容：

```
config sambashare
        option 'name'
        option 'path'
        option 'read_only'        'no'
        option 'guest_ok'         'yes'
        option 'create_mask'              '0777'
        option 'dir_mask'         '0777'
        #option 'users'                   'abc'
```

图 9.3- 1/etc/config/samba 需添加的内容

按照上面的格式修改添加在第一个文件中，把第一行 option ‘name’ 后面的名字改成自己想共享的文件夹名，第二行 option ‘path’ 改成文件夹所在的路径即可，重新上电之后就可以在\\192.168.1.1 里看到自己想共享的文件夹了。

也可以用手机通过 es 文件浏览器来查看共享文件夹的内容，首先手机里要确定安装 es 文件浏览器，然后手机 wifi 连接上我们的开发板，打开手机的 es 浏览器，如图 9.3- 2 所示：



图 9.3- 2es 浏览器

点击新建出现图 9.3- 3：

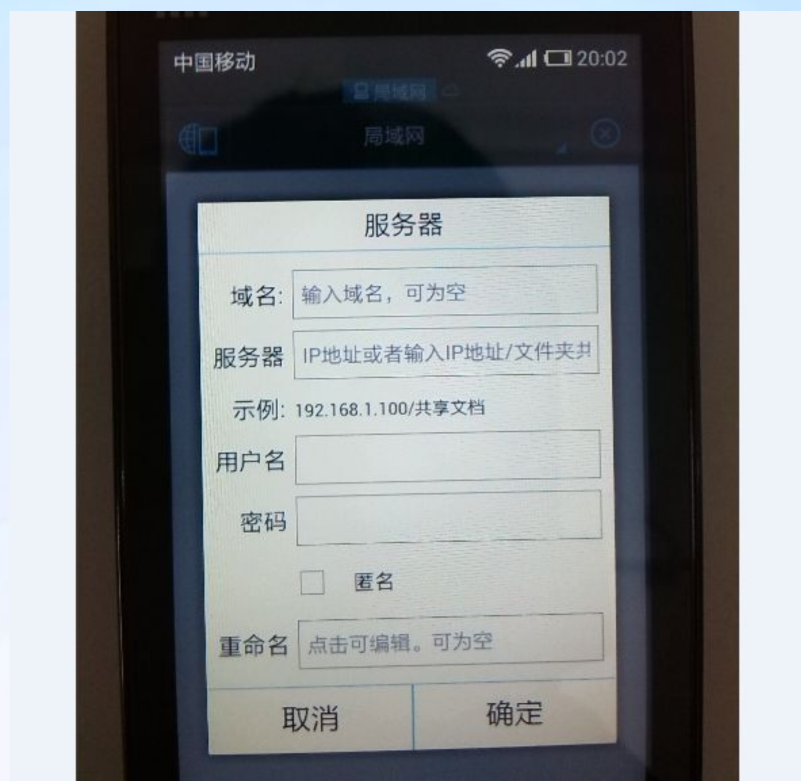


图 9.3- 3 输入 IP 等信息的界面

输入 IP，用户名和密码（192.168.1.1，用户名是 nobody，密码是刚才就修改的）：

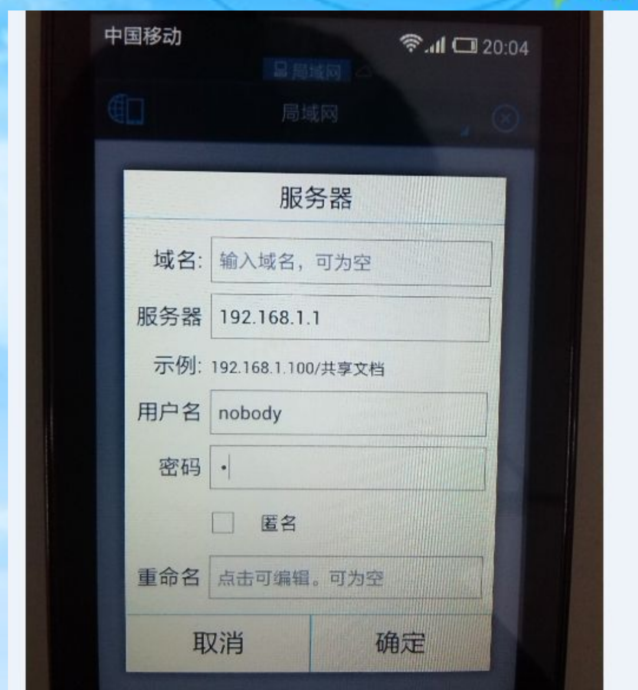


图 9.3- 4 已配置好的界面

然后点击确定，就可以看到共享文件夹了。如图 9.3- 5 所示：



图 9.3- 5 共享文件夹内容

第 21 章 网络流媒体服务构建与应用

开发板自带的 madplay 软件可支持 MP3 格式的音频播放，我们只需将 USB 声卡连接到开发板上，输入一行命令，便可以播放音频了。

首先，将声卡连接至开发板，插入耳机，在 SecureCRT 终端里会看到如图 10.1-1 所示的信息：

```
root@MicroCloud:/tmp# [ 3507.830000] usb 1-1.4.2: new full-speed USB device num
ber 10 using ehci-platform
[ 3507.970000] usb 1-1.4.2: no of_node; not parsing pinctrl DT
[ 3507.990000] snd-usb-audio 1-1.4.2:1.0: no of_node; not parsing pinctrl DT
[ 3508.070000] usbhid 1-1.4.2:1.3: no of_node; not parsing pinctrl DT
```

图 10.1- 1 声卡显示信息

用 winSCP 将音频文件上传至开发板中，输入命令：

madplay 音频文件名

就可以听到音频的播放了：

```
root@MicroCloud:/tmp# madplay 123.mp3
MPEG Audio Decoder 0.15.2 (beta) - Copyright (C) 2000-2004 Robert Leslie et al.
█
```

图 10.1- 2 播放音频

注意如果当前文件夹和音频文件不在同一目录下的话，音频文件名的位置要写绝对路径，比如 madplay /tmp/123.mp3。下面是几个 madplay 的常用命令（&表示后台运行）：

- madplay north.mp3 & 调用 madplay 播放器播放*.mp3 音乐
- madplay north.mp3 -r &循环播放：参数-r
- killall -9 madplay 利用 system 函数调用 killall 命令将 madplay 终止掉
- killall -STOP madplay &利用 system 函数调用 killall 命令将 madplay 暂停
- killall -CONT madplay &利用 system 函数调用 killall 命令恢复 madplay 的播放

STM32+LINUX 开发板实战篇

第 22 章 嵌入式 Linux 串口与网络编程基础

22.1 概述

这一章我们主要介绍了 Linux 下串口与网络的编程，包括如何操作串口以及网络编程中所涉及的函数功能、串口的读写和网络收发 UDP 包的详细流程，分析了应用程序执行的过程。

该应用程序只包括一个 udp_serial.c 文件，在 Linux 下交叉编译出可执行文件（文件的名字可以任意取）。将可执行文件用 WinSCP 工具上传到开发板系统中然后运行即可。

此应用程序的功能包括两个部分：一是程序将串口接收过来的数据，以 UDP 数据包的形式发送到网络中去；二是从网络中接收 UDP 广播包，提取出有效数据，将数据用串口发送出去。程序可以接收 UDP 广播包，或者定点包。能够向外发送 UDP 广播包，在同一网段的主机都可以接收到该程序发送的 UDP 数据。

此应用程序的两部分功能是并行执行的，因此我们要创建两个进程，让两个功能能同时进行，如下图 11.1- 1 所示：

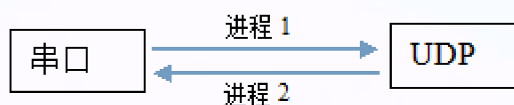


图 11.1- 1 串口和 UDP 的互相转换

22.2 相关知识介绍

22.2.1 Linux 下串口编程简介

1）Linux 下串口

Windows 下的串口通信相信大家已经很熟悉了，因此，这里我们只针对 Linux 下的串口应用进行介绍。

Linux 操作系统将所有的设备看成文件，以操作文件的方式访问设备。所有的外部设备都以文件的形式存放在/dev 目录下，/dev 目录下的内容如图 11.2- 1：

```
root@MicroCloud:/dev# ls
audio              mtd0          mtdblock1             sda1
autofs             mtd0ro        mtdblock2             sda2
bus                mtd1          mtdblock3             sg0
console            mtd1ro        mtdblock4             shm
cpu_dma_latency    mtd2          mtdblock5             snd
dsp                mtd2ro        network_latency       tty
etherd             mtd3          network_throughput    ttyS0
full               mtd3ro        null                  ttyS1
input              mtd4          ppp                   urandom
kmsg               mtd4ro        ptmx                  video0
log                mtd5          pts                   watchdog
mem                mtd5ro        random                watchdog0
mixer              mtdblock0     sda                   zero
```

图 11.2- 1/dev 目录下的内容

上图即为/dev 目录下的所有设备，其中有两个设备 ttyS0，ttyS1，就是我们所要学习的串口，可见，我们的开发板支持两个串口，其中 ttyS0 是我们连接的超级终端。为了便于理解，我们将 Windows 下的串口名称与 Linux 下的串口名称作了类比，类比结果如图 11.2- 2：

操作系统	串口 1	串口 2	USB/RS-232 转换器
Windows	COM1	COM2	-
Linux	/dev/ttyS0	/dev/ttyS1	/dev/ttyUSB0

图 11.2- 2Windows 下与 Linux 下的串口名称类比

接下来我们将详细介绍如何对串口进行编程。

2）串口操作需要的头文件

串口操作需要包含的头文件如图 11.2- 3 所示：

```
#include     <stdio.h>        /*标准输入输出定义*/
#include     <stdlib.h>       /*标准函数库定义*/
#include     <unistd.h>       /*Unix 标准函数定义*/
#include     <sys/types.h>
#include     <sys/stat.h>
#include     <fcntl.h>        /*文件控制定义*/
#include     <termios.h>      /*PPSIX 终端控制定义*/
#include     <errno.h>        /*错误号定义*/
```

图 11.2- 3 串口操作需要包含的头文件

3）串口的打开

相信学过 C 语言的同学都知道如何打开一个文件，打开串口就像打开一个文件一样，通过使用标准的文件打开函数操作，操作过程如图 11.2- 4 所示：

```
int fd;
/*以读写方式打开串口*/
fd = open( "/dev/ttyS0", O_RDWR);
if (-1 == fd){
/* 不能打开串口一*/
perror(" 提示错误! ");
}
```

图 11.2- 4 打开串口

图 11.2- 4 中，open()函数中 O_RDWR 的意思是以可读可写的方式打开串口，除了这种方式之外，还包括以下几种方式：

O_RDONLY ------->以只读方式打开

O_WRONLY ------->以只写方式打开

O_NOCTTY -------->如果 pathname 指的是终端设备，则不将此设备分配作为此进程的控制终端。

O_NONBLOCK 如果 pathname 指的是一个 FIFO、一个块特殊文件或一个字符特殊文件，则此选择项为此文件的本次打开操作和后续的 I / O 操作设置非阻塞方式。

当然，还有其他选项的定义，限于篇幅，这里就不一一介绍了。

打开一个串口之后，open()函数会返回一个句柄 fd，接下来我们对串口的操作，实际上都是对句柄的操作。返回值 fd:串口打开成功则返回文件描述符，如果失败返回-1；

4）串口的配置

配置串口主要是配置串口的传输速率，校验位和停止位等。图 11.2- 5 是设置波特率，也就是传输速率的函数。

```
/**
*@brief  设置串口通信速率
*@param  fd     类型 int  打开串口的文件句柄
*@param  speed  类型 int  串口速度
*@return  void
*/
int speed_arr[] = { B38400, B19200, B9600, B4800, B2400, B1200, B300,
          B38400, B19200, B9600, B4800, B2400, B1200, B300, };
int name_arr[] = {38400,  19200,  9600,  4800,  2400,  1200,  300, 38400,
          19200,  9600, 4800, 2400, 1200,  300, };
void set_speed(int fd, int speed){
  int   i;
  int   status;
  struct termios   Opt;
  tcgetattr(fd, &Opt);
  for ( i= 0;  i < sizeof(speed_arr) / sizeof(int);  i++) {
    if  (speed == name_arr[i]) {
      tcflush(fd, TCIOFLUSH);
      cfsetispeed(&Opt, speed_arr[i]);
      cfsetospeed(&Opt, speed_arr[i]);
      status = tcsetattr(fd1, TCSANOW, &Opt);
      if  (status != 0) {
        perror("tcsetattr fd1");
        return;
      }
      tcflush(fd,TCIOFLUSH);
    }
  }
}
```

图 11.2- 5 设置波特率的函数

接下来是设置校验停止位的函数：

```
/**
*@brief   设置串口数据位，停止位和效验位
*@param  fd     类型  int  打开的串口文件句柄
*@param  databits 类型  int 数据位   取值 为 7 或者8
*@param  stopbits 类型  int 停止位   取值为 1 或者2
*@param  parity  类型  int  效验类型 取值为N,E,O,,S
*/
int set_Parity(int fd,int databits,int stopbits,int parity)
{
        struct termios options;
        if  ( tcgetattr( fd,&options)  !=  0) {
                perror("SetupSerial 1");
                return(FALSE);
        }
        options.c_cflag &= ~CSIZE;
        switch (databits) /*设置数据位数*/
        {
        case 7:
                options.c_cflag |= CS7;
                break;
        case 8:
                options.c_cflag |= CS8;
                break;
        default:
                fprintf(stderr,"Unsupported data size\n"); return (FALSE);
        }
switch (parity)
{
        case 'n':
        case 'N':
                options.c_cflag &= ~PARENB;   /* Clear parity enable */
                options.c_iflag &= ~INPCK;     /* Enable parity checking */
                break;
        case 'o':
        case 'O':
                options.c_cflag |= (PARODD | PARENB); /* 设置为奇效验*/
                options.c_iflag |= INPCK;             /* Disnable parity checking */
                break;
        case 'e':
        case 'E':
                options.c_cflag |= PARENB;     /* Enable parity */
                options.c_cflag &= ~PARODD;   /* 转换为偶效验*/
                options.c_iflag |= INPCK;       /* Disnable parity checking */
                break;
        case 'S':
        case 's':  /*as no parity*/
            options.c_cflag &= ~PARENB;
                options.c_cflag &= ~CSTOPB;break;
        default:
                fprintf(stderr,"Unsupported parity\n");
                return (FALSE);
        }
```

图 11.2- 6 设置停止位等 1




```
/* 设置停止位*/
switch (stopbits)
{
        case 1:
                options.c_cflag &= ~CSTOPB;
                break;
        case 2:
                options.c_cflag |= CSTOPB;
           break;
        default:
                 fprintf(stderr,"Unsupported stop bits\n");
                 return (FALSE);
}
/* Set input parity option */
if (parity != 'n')
        options.c_iflag |= INPCK;
tcflush(fd,TCIFLUSH);
options.c_cc[VTIME] = 150; /* 设置超时15 seconds*/
options.c_cc[VMIN] = 0; /* Update the options and do it NOW */
if (tcsetattr(fd,TCSANOW,&options) != 0)
{
        perror("SetupSerial 3");
        return (FALSE);
}
return (TRUE);
}
```

图 11.2- 7 设置停止位等 2

5）串口的读和写

发送串口数据(即向串口写入数据),用 write()函数实现，具体语句如下：

```
char  buffer[1024];int   Length;int   nByte;nByte = write(fd, buffer ,Length)
```

上述语句中 write()函数实现的功能是：将以 buffer 指针所指地址为首地址，长度为 Length 个字节的数据写到句柄 fd 所指的串口中去。
接收串口数据(即从串口读取数据)，用 read()函数实现，具体语句如下：

```
char  buff[1024];int   Len;int  readByte = read(fd,buff,Len);
```

上述语句中 read()函数实现的功能是从 fd 所指的串口中取出 len 个字节放到以 buff 为首地址的内存中。readByte 是实际读取的字节数。
串口的读写模式有直接模式和缓存模式，我们这里用的是缓存模式，实际上每次读取串口的字节数都是不固定的，所以通信过程中无法保证一次发送的数据能一次性接收。为此，我们将采用一个 while（）循环来保证接收数据的完整性。这个在后面会提到。

6）串口的关闭

关闭串口就是关闭文件，用下面的语句实现：

close（fd）;

22.2.2 Linux 下 UDP 网络编程简介

Linux 下网络编程是一个很庞大的知识架构，对此有专门的书籍介绍，这里我们只介绍一下和 UDP 传输有关的网络知识。

1）进程

在 Linux 中，进程是正在执行的程序。它相当于 Windows 环境内的任务这一概念。每个进程包括程序代码和数据。其中数据包含程序变量数据、外部数据和程序堆栈等。系统的命令解释程序 shell 为了执行一条命令，就要建立一个新的进程并运行它，例如：

$cat file1

该命令就会使 shell 专门建立一个进程来运行 cat 命令。

再看一个复杂一些的命令：

$ls | wc –ll

这个命令就会使 shell 建立两个进程，以并行执行命令 ls 和 wc,把查询目录列表命令 ls 的输出通过管道送至字计数命令 wc。一个进程对应于一个程序的执行，因此绝对不要把进程与程序这两个概念相混淆。

进程是动态的，而程序为静态的。实际上，多个进程可以并发执行同一个程序，对于公用的实用程序就常常是这样。例如，几个用户可以同时运行一个编辑程序，每个用户对此程序的执行均是一个单独的进程。

在 Linux 中，一个进程又可以启动另一个进程，比如我们实现串口与网络编程的 C 语言程序的执行，就是一个进程，在这个 C 语言程序中，还需要同时完成两项任务，一是接收 UDP 数据包，并用串口发送出去，二是接收串口的数据，利用 UDP 协议将其打包成 UDP 包发送出去。而这两个任务的完成又需要启动两个进程。

2）fork()函数

系统调用 fork()是建立进程的最基本操作，它是把 Linux 变换为多任务系统的基础。fork()在 Linux 系统库 unistd.h 中的函数声明如下：

pid_t fork(void);

如果 fork()调用成功，就会使内核建立一个新的进程，所建的新进程是调用 fork()的进程的副本。

新建立的进程被称为子进程（child process），那个调用 fork()建立此新进程的进程被称为父进程（parent process）。以后，父进程与子进程就并发执行，它们都从 fork()调用后的那句语句开始执行。图 11.2- 8 给出了调用 fork()的情况：

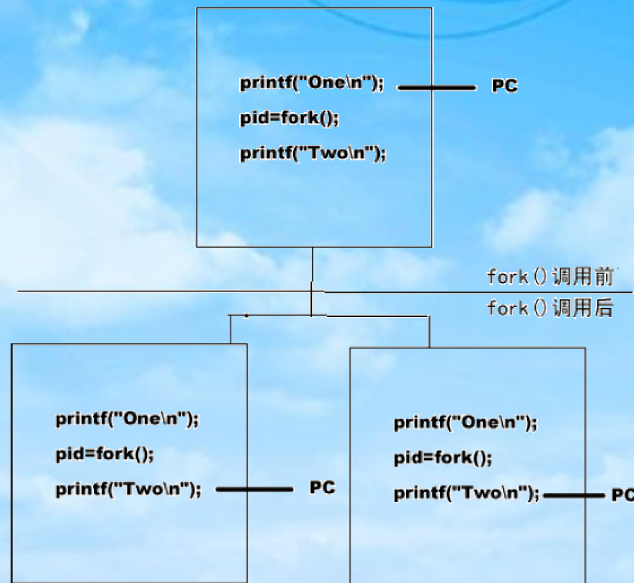


图 11.2- 8 调用 fork()

它分为 fork()调用前和调用后两部分。调用前的那一部分给出了进程 A 调用 fork()的情况。PC（程序计数器）指向当前执行的语句。这时它指向第一个 printf 语句。调用后那一部分给出了调用 fork()以后的情况。这时进程 A 和 B 一起运行，进程 A 是父进程，进程 B 是子进程，它是进程 A 的副本，执行与 A 一样的程序。两个 PC 都指向第二个 printf 语句，即 fork()调用之后的语句。也就是说，A 和 B 都从程序的相同点开始执行。

系统调用 fork()没有参数，它返回一个 pid_t 类型的值 pid。pid 被用来区分父进程和子进程。在父进程中，pid 被置为一个非 0 的正整数；在子进程中，pid 被置为 0。根据 fork()在父进程和子进程中的返回值不同，程序员可以据此为两个进程指定不同的工作。在父进程中，pid 中返回的数是子进程的进程标识符。这个数用于在系统中表示一个进程，就像用户标识符标识一个用户那样。因为所有的进程都是通过 fork()调用形成的，所以每个 Linux 进程都有自己的进程标识符，而且它是唯一的。

3）socket 套接字

作为 Linux 系统的进程通信机制，socket 如同插座一样方便的帮助计算机接入互联网通信，随后的 UDP 的连接建立、数据传输等操作都是通过该 socket 实现的。常用的 socket 类型有三种：流式套接字（SOCK_STREAM），数据报套接字（SOCK_DGRAM）及原始套接字（SOCK_RAM）。因为我们是 UDP 传输，所以采用的是面向无连接的数据报套接字（SOCK_DGRAM）。

我们通过调用 socket 函数，来完成对 socket 的建立，该函数会返回一个类似于文件描述符的句柄。我们可以将这个描述符看成普通的文件的描述符来操作。socket 函数的定义是下面这样子的：

```
#include <sys/types.h>
#include <sys/socket.h>
int socket（int domain , int type , int protocol）;
```



首先，domain 指明所使用的协议族，domain 需要被设置为“AF_INET”，就像上面的 struct sockaddr_in。然后，type 参数告诉内核这个 socket 是什么类型，“SOCK_STREAM”或是“SOCK_DGRAM”。我们需要设置为 SOCK_DGRAM。protocol 通常赋值为“0”

4）bind()函数

在使用 socket 进行网络传输以前，必须配置 socket，通过 bind()函数来配置本地信息。

bind()的系统调用声明如下：

```
#include <sys/types.h>
#include <sys/socket.h>
int bind (int sockfd , struct sockaddr *my_addr , int addrlen) ;
```

参数说明：

sockfd 是由 socket()函数返回的套接字描述符。addrlen 是 sockaddr 结构的长度，可以设置为 sizeof(struct sockaddr)。my_addr 指定一个 sockaddr 结构，该结构可以是这样定义的：

```
struct   sockaddr_in{
       short   sin_family;       /*地址族*/
       u_short   sin_port;
       struct   in_addr   sin_addr;
       char    sin_zero;
}
```

其中 sin_family 置 AF_INET；sin_port 指明端口号；sin_addr 结构体中只有唯一的字段 s_addr，表示 IP 地址，一般用函数 inet_addr()把字符串形式的 IP 地址转换成 unsigned long 的整数值后再置给 s_addr。

5）setsockopt() 和 getsockopt() 函数

Linux 所提供的 socket 库含有一个错误（bug）。此错误表现为你不能为一个套接字重新启用同一个端口号，即使在你正常关闭该套接字以后。例如，比方说，你编写一个服务器在一个套接字上等待的程序．服务器打开套接字并在其上侦听是没有问题的。无论如何，总有一些原因（不管是正常还是非正常的结束程序）使你的程序需要重新启动。然而重启动后你就不能把它绑定在原来那个端口上了。从 bind()系统调用返回的错误代码总是报告说你试图连接的端口已经被别的进程所绑定。问题就是 Linux 内核在一个绑定套接字的进程结束后从不把端口标记为未用。在大多数 Linux/UNIX 系统中，端口可以被一个进程重复使用，甚至可以被其它进程使用。

在 Linux 中绕开这个问题的办法是，当套接字已经打开但尚未有连接的时候用 setsockopt()系统调用在其上设定选项（options）。setsockopt() 调用设置选项而 getsockopt()从给定的套接字取得选项。

这里是这些调用的语法：

```
#include<sys/types.h>
#include<sys/socket.h>
int getsockopt(int sockfd, int level, int name, char *value, int *optlen);
int setsockopt(int sockfd, int level, int name, char *value, int *optlen);
```

下面是两个调用的参数说明：

sockfd 必须是一个已打开的套接字。

level 是函数所使用的协议标准（protocol level）（TCP/IP 协议使用 IPPROTO_TCP，套接字标准的选项实用 SOL_SOCKET）。

name 选项在套接字说明书中（man page）有详细说明。

value 指向为 getsockopt()函数所获取的值，setsockopt()函数所设置的值的地址。

optlen 指针指向一个整数，该整数包含参数以字节计算的长度。

现在我们再回到 Linux 的错误上来．当你打开一个套接字时必须同时用下面的代码段来调用 setsockopt()函数：

```
/* 设定参数数值 */
opt = 1; len = sizeof(opt);
/* 设置套接字属性 */
setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&opt,&len);
```

setsockopt()函数还有很多其他用法，请参考帮助页（man pages）.

6）sendto() 和 recvfrom() 函数

sendto() 和 recvfrom() 函数是进行无连接的 UDP 通讯时使用的。使用这两个函数，则数据会在没有建立过任何连接的网络上传输。因为数据报套接字无法对远程主机进行连接，在发送数据前需要知道远程主机的 IP 地址和端口。

下面是 sendto()函数的声明：

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto（int sockfd, const void *msg, int len, unsigned int flags,
const struct sockaddr *to, int tolen）;
```

sockfd 是代表你与远程程序连接的套接字描述符。

msg 是一个指针，指向你想发送的信息的地址。

len 是你想发送信息的长度。

flags 发送标记。一般都设为 0。（你可以查看 send 的 man pages 来获得其他的参数值并且明白各个参数所代表的含义）

to 是一个指向 struct sockaddr 结构的指针，里面包含了远程主机的 IP 地址和端口数据。

tolen 只是指出了 struct sockaddr 在内存中的大小 sizeof(struct sockaddr)。

sendto()返回它所真正发送的字节数，当它发生错误的时候，返回 –1 ，同时全局变量 errno 存储了错误代码。

recvfrom()的声明为：

```
#include <sys/types.h>
#include <sys/socket.h>
int recvfrom(int sockfd, void *buf, int len, unsigned int flags
```



struct sockaddr *from, int *fromlen);
其参数含义如下：
sockfd 是你要读取数据的套接字描述符。
buf 是一个指针，指向你能存储数据的内存缓存区域。
len 是缓存区的最大尺寸。
flags 是 recv() 函数的一个标志，一般都为 0
from 是一个本地指针，指向一个 struct sockaddr 的结构(里面存有源 IP 地址和端口数).
fromlen 是一个指向一个 int 型数据的指针，它的大小应该是 sizeof (struct Sockaddr). 当函数返回的时候，fromlen 指向的数据是 from 指向的 struct sockaddr 的实际大小.
recvfrom() 返回它接收到的字节数，如果发生了错误，它就返回–1 ，全局变量 errno 存储了错误代码. 如果一个信息大得缓冲区都放不下，那么附加信息将被砍掉。该调用可以立即返回,也可以永久的等待。这取决于你把 flags 设置成什么类型。你甚至可以设置超时(timeout)值。

22.3 应用程序代码分析

首先是头文件，程序需要包含图 11.3- 1 所示头文件：

```
#include<stdlib.h>
#include<stdio.h>
#include<errno.h>
#include<string.h>
#include<netdb.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<termios.h>
#include<sys/wait.h>
#define PORT 61557              //远端主机端口号
#define BROADCASTPORT 8003      //本地端口号
#define BUF_SIZE 100            //定义UDP数据发送的长度
#define FALSE  -1
#define TRUE   0
```

图 11.3- 1 头文件

上图中除了包含需要的头文件，还定义了 UDP 协议的远程主机端口号及本地端口号。
接下来是波特率、校验位和停止位的配置：




```
/**
*@bref 设置串口通信速率
*@param fd 类型 int 打开串口文件的句柄
*@param speed 类型 int 串口速度
*@return void
*/
int speed_arr[] = {B115200, B38400, B19200, B9600, B4800, B2400, B1200, B300,
        B38400, B19200, B9600, B4800, B2400, B1200, B300, };
int name_arr[] = {115200,38400,  19200,  9600,  4800,  2400,  1200,  300,
 38400, 19200,  9600, 4800, 2400, 1200,  300, };
void set_speed(int fd, int speed){
  int   i;
  int   status;
  struct termios   Opt;
  tcgetattr(fd, &Opt);
  for ( i= 0;  i < sizeof(speed_arr) / sizeof(int);  i++) {
    if  (speed == name_arr[i]) {
      tcflush(fd, TCIOFLUSH);
      cfsetispeed(&Opt, speed_arr[i]);
      cfsetospeed(&Opt, speed_arr[i]);
      status = tcsetattr(fd, TCSANOW, &Opt);
      if  (status != 0) {
        perror("tcsetattr fd");
        return;
      }
      tcflush(fd,TCIOFLUSH);
    }
  }
}
/**
*@bref 设置串口数据位，停止位和校验位
*@param fd 类型 int 打开串口文件的句柄
*@param databits 类型 int 数据位 取值为7或8
*@param stopbits 类型 int 停止位 取值为1或2
*@param parity 类型 int 校验位 取值为N，E，O，S
*@return void
*/
```




```
int set_Parity(int fd,int databits,int stopbits,int parity)
{
    struct termios options;
    if  ( tcgetattr( fd,&options)  !=  0) {
        perror("SetupSerial 1");
        return(FALSE);
    }
    options.c_cflag &= ~CSIZE;
    options.c_iflag &= ~INPCK;
    options.c_iflag |= IGNBRK;
    options.c_iflag &= ~ICRNL;
    options.c_iflag &= ~IXON;
    options.c_lflag &= ~IEXTEN;
    options.c_lflag &= ~ECHOK;
    options.c_lflag &= ~ECHOCTL;
    options.c_lflag &= ~ECHOKE;
    options.c_oflag &= ~ONLCR;
    switch (databits) /*设置数据位*/
    {
    case 7:
        options.c_cflag |= CS7;
        break;
    case 8:
        options.c_cflag |= CS8;
        break;
    default:
//      fprintf(stderr,"Unsupported data size\n");
                return (FALSE);
    }
```




```
switch (parity)
{
    case 'n':
    case 'N':
        //options.c_cflag &= ~PARENB;   /* Clear parity enable */
        //options.c_iflag &= ~INPCK;     /* Enable parity checking */
        options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
        options.c_oflag &= ~OPOST; /*Output*/
        break;
    case 'o':
    case 'O':
        options.c_cflag |= (PARODD | PARENB); /* 设置奇校验*/
        options.c_iflag |= INPCK;             /* Disnable parity checking */
        break;
    case 'e':
    case 'E':
        options.c_cflag |= PARENB;     /* Enable parity */
        options.c_cflag &= ~PARODD;   /* 转换为偶校验*/
        options.c_iflag |= INPCK;     /* Disnable parity checking */
        break;
    case 'S':
    case 's':  /*as no parity*/
        options.c_cflag &= ~PARENB;
        options.c_cflag &= ~CSTOPB;break;
    default:
//      fprintf(stderr,"Unsupported parity\n");
        return (FALSE);
    }
/* 设置停止位*/
switch (stopbits)
{
    case 1:
        options.c_cflag &= ~CSTOPB;
        break;
    case 2:
        options.c_cflag |= CSTOPB;
       break;
    default:
//       fprintf(stderr,"Unsupported stop bits\n");
        return (FALSE);
}
/* Set input parity option */
if ((parity != 'n')&&(parity != 'N'))
    options.c_iflag |= INPCK;
tcflush(fd,TCIFLUSH);
options.c_cc[VTIME] = 150; /* 设置超时1 seconds*/
options.c_cc[VMIN] = 0; /* Update the options and do it NOW */
if (tcsetattr(fd,TCSANOW,&options) != 0)
{
//  perror("SetupSerial 3");
    return (FALSE);
}
return (TRUE);
}
```

图 11.3- 2 串口波特率、校验等设置



完成了串口的相关配置后，接下来以读写方式打开串口(IO 口为非阻塞模式)：

```
/**
*@bref 打开串口
*@param Dev 类型 char 串口号
*@return int 返回串口的句柄
*/
int OpenDev(char *Dev)
{
    //以读写方式打开串口，IO操作为非阻塞模式
    int fd = open( Dev, O_RDWR|O_NOCTTY|O_NONBLOCK );
    if (-1 == fd)
    {
//      perror("Can't Open Serial Port");
        return -1;                    //若打开失败，返回-1
    }
    else
        return fd;
}
```

图 11.3- 3 打开串口函数

下面是从外界接收 UDP 数据包的函数：



```c
int udp_broadcast_service(unsigned char *change)
{
    int castsockfd, connfd;
    int optval,client_len,revc;
    struct sockaddr_in serv_addr, client_addr;
    if ( (castsockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 )//创建socket
      {
        exit(1);
      }
//配置本地信息，包括端口号，IP，和协议族（一般为AF_NET,表示UNIX网络套接字）
//把serv_addr的所有字节设置成字符
    memset( &serv_addr, 0, sizeof(struct sockaddr_in) );
    serv_addr.sin_family = AF_INET;   //表示UNIX网络套接字
    //设置本地端口号 htons()表示把16位从主机字节序转换成网络字节序
    serv_addr.sin_port = htons( BROADCASTPORT );
    //INADDR_ANY使用自己的IP地址
    serv_addr.sin_addr.s_addr = htons( INADDR_ANY );
    fcntl(castsockfd,F_SETFL,O_NONBLOCK);
    //调用bind函数将本地信息与socket相关连
    if ( bind(castsockfd, (struct sockaddr *)&serv_addr,
              sizeof(struct sockaddr)) < 0 )
      {
        exit(1);
      }
    client_len = sizeof( struct sockaddr_in );
    while((revc=recvfrom(castsockfd, change, BUF_SIZE, 0, //接收UDP数据包
                (struct sockaddr *)&client_addr,
                 &client_len))<=0)
        {
          if(getppid()==1)             //父进程死亡，子进程退出
          exit(1);
        }
    close( castsockfd );                        //关闭套接字
    return revc;                                //返回数据长度
}
```

图 11.3- 4 从外界接收 UDP 数据包

第一步是创建套接字，castsockfd 是套接字描述符。

第二步是配置本地信息。因为我们要接收 UDP 数据包，所以在存放本地信息的结构体 serv_addr 中，端口号和 IP 都要设置成本地的，这里定义的本地端口号为 8003(在 0~65535 的所有端口号中，0~1024 是系统的端口号，因此我们可以再 1025~65535 区间内任意设置）。INADDR_ANY 表示使用自己的 IP 地址，memset()函数是把 serv_addr 的所有字节设置成字符，hton()设置本地端口号 htons()表示把 16 位从主机字节序转换成网络字节序。

第三步是用 bind()函数将本地信息与套接字绑定。

第四步是接收 UDP 数据包。数据存放到 change 所指的首地址内存中。

最后关闭套接字，返回数据长度。

接下来是发送 UDP 数据包的函数：

```c
int send_broadcast(unsigned char *send_buf,int t)
{
    int client_sockfd;
    int optval;
    struct sockaddr_in serv_addr;
    if ((client_sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)//socket
        {
//            perror("sockfd error!\n");
            exit(1);
        }
    optval = 1;
//当一个套接字已经打开但尚未有连接的时候用setsockopt()系统调用在其上设定选项
    if ( setsockopt(client_sockfd, SOL_SOCKET, SO_BROADCAST,
                    (void *)&optval, sizeof(int)) < 0 )
       {
//            perror("setsockopt error!\n");
            exit(1);
       }
       //把serv_addr的所有字节设置成字符
    memset( &serv_addr, 0, sizeof(struct sockaddr_in) );
    serv_addr.sin_family = AF_INET;          //表示UNIX网络套接字
    serv_addr.sin_port = htons( PORT );       //远端主机端口号
    serv_addr.sin_addr.s_addr = inet_addr("192.168.1.255");//发送的是广播包
    if ( sendto(client_sockfd, send_buf, t, 0,
                (struct sockaddr *)&serv_addr,
                 sizeof(struct sockaddr)) < 0 )//广播包每次发一个字节
        {
//            perror("send failed!\n");
            exit(1);
        }
    close( client_sockfd );                    //关闭套接字
    return 0;
}
```

图 11.3- 5 发送 UDP 数据包

第一步是创建套接字，client_sockfd 是套接字描述符。

第二步是配置远端主机信息。因为我们要发送 UDP 数据包，所以，在存放远端主机信息的结构体 serv_addr 中，端口号，和 IP 都要设置成对方的，这里我们的端口号是 61557。因为我们发送广播包，所以地址设置为广播地址。因为在另一个进程我们已经打开的这个套接字，这时候 bind()函数就配置了，所以我们调用了 setsockopt()函数，来设定选项，关于本函数的解释请看第二章。hton()设置对方端口号 htons()表示把 16 位从主机字节序转换成网络字节序。

第三步是将 send_buff 所指的首地址的数据，用 UDP 数据包发送出去。

最后关闭套接字，返回 0。

接下来我们来讲解一下主函数，为了能讲的更清晰一些，我们将主函数分开解说：

```
int main()
{
        pid_t pid,id;
    int fd;
    int nread,nwrite;
        int i=0,sum=0,recv_c=0;
    unsigned char readbuff[100];      //定义串口在内存中接收的缓冲区
        unsigned char buff[100];  //将缓冲接收的串口数据放到buff里
        unsigned char check[100];   //存放udp接收的数据
    char *dev  = "/dev/ttyS0";
    fd = OpenDev(dev);                //打开串口
    set_speed(fd,57600);         //设置波特率为9600
    if (set_Parity(fd,8,1,'N') == FALSE)    //设置校验，数据位，停止位
            {
        exit (1);
       }
```

图 11.3- 6 主函数 1

图 11.3- 6 这部分函数的主要功能就是建立发送和接收数据缓冲区，定义变量，以及打开、设置串口。接下来是创建进程：

```
        pid=fork();     //建立新的进程，以下是两个并行的进程
```

fock()函数用于创建进程(上一节已经提过)函数返回 pid_t 类型的值 pid，我们根据 pid 的值来区分子进程和父进程。这里比较难理解，因为通常的变量同一时刻只能存储一个值，而 pid 则不同，调用了此函数后，pid 不是一个值，而是两个值，一个值为 0，是子进程的标号，另一个大于 0，是父进程的标号。若 pid<0，则说明当前没有进程，fock()函数创建进程失败。若 pid=0，这时执行的是子进程的程序，由图 11.3- 7 可以看出，子进程的功能就是接收 UDP 数据，用串口发送出去。

```
        if(pid<0)
            {
                exit(1);
            }

        if(pid==0)             //子进程，接收广播包，发送到串口
            {
//              printf( "child\n");
                while(1)
                {
                  recv_c=udp_broadcast_service(check);     //接收udp数据
                  //将数据通过串口发送出去
                  while((nwrite=write(fd,check,recv_c))>0)
                        {
                  check[nwrite] = '\0';
                        break;
                        }
                }
            }
```

图 11.3- 7 主函数 2

否则，pid>0 时，这时执行的是父进程。其功能是接收串口的数据，将其以 UDP 数据包形式发送到网络。具体实现过程如图 11.3- 8 所示：

```
        else                    //父进程，接收串口的数据，发送广播包
            {
//              printf( "parent\n");
            while(1)
             {
                tcflush(fd, TCIOFLUSH);    //刷新串口
                bzero(readbuff,100);       //清空缓冲区
                bzero(buff,100);
                while((nread = read(fd, buff,1))<=0)
                    {
      //等待子进程，如果子进程死亡，返回子进程pid，如果没死亡返回pid=0;
                        id=waitpid(pid,NULL,WNOHANG);
                        if(id==pid)//如果子进程死亡，父进程销毁
                          {
//                            printf("child exit\n");
                          exit(1);
                          }
                    usleep(1000);
                    }
                sum=1;
                nread=0;
          usleep(500000);                    //要让串口有足够的时间接数
//                  printf( "parent\n");
                while((nread = read(fd, readbuff,8))>0)//接收串口的数据
                 {

                    for(i=0;i<nread;i++)
                     {
                        buff[sum+i]=readbuff[i];
                     }
            sum=nread+sum;
            nread=0;
            bzero(readbuff,100);
                    usleep(10000);
//          printf( "sum=%d\n", sum);
```

图 11.3- 8 主函数 3

值得注意的是串口接收函数 read()，虽然我们在程序里写的是每次接收 8 个字节

```
while((nread = read(fd, readbuff,8))>0)  //接收串口的数据
```

但是实际上串口每次实际接收的字节数不一定是正好 8 个，有时这 8 个字节分两次接收到，比如一次接收 3 个，第二次接收 5 个；或者是每次接收 4 个，共接两次。这是由于串口本身的缓存机制决定的，所以我们就设置了一个 sum 的变量，当接收完第一个字节时，sum 为 1，这时 buff 里已经有一个字节了：

```
while((nread = read(fd, buff,1))<=0)
```

buff[sum+i]=readbuff[i];若此时刚接收到 1 个字节时，此句是将新接收的数据按顺序继续存放到 buff[1]以后的数组里；若此时收到多个字节时，sum 变量则为之前收到的所有数据之和(sum=nread+sum;)，buff[sum+i]=readbuff[i]则表示将 readbuff 数组，也就是新接收的数据，继续放到 buff 数组中第 sum 个位置以后的数组中。直到串口的缓冲区没有数据，则跳出 while()循环，这样就能保证完全接收到串

口的数据啦。

至此，我们就将这个程序分析完了。

22.4 应用程序执行

应用程序实现的功能是 UDP 转串口通信，所以需要一台上位机并且可以和开发板保持正常的网络通信(有线，无线或 3G)并配置 IP 为 192.168.1.xxx 保证上位机和开发板在同一网段，上位机提供一个可以发送和接收 UDP 数据包的调试软件，这里用的是一款非常好用的网络和串口调试工具 USR-TCP232-Test，在资料包中的“所用到的工具”里可以找到。

22.4.1 运行前相关配置

我们将 Linux 下编译出的可执行文件命名为 udp_serial.o，用 WinSCP 将文件上传到开发板的 Program 文件夹下，（如何编译和下载可执行文件请参看第二部分第二章），修改下文件的权限。

```
root@MicroCloud:/# cd Program/
root@MicroCloud:/Program# ls
helloworld.o  ser2web       seril_udp     udp_serial.o
root@MicroCloud:/Program#
```

可以看到有一个 udp_serial.o 的文件，输入运行命令

```
root@MicroCloud:/Program# ./udp_serial.o
```

当显示上图内容时，表示程序已经运行，然后我们把 SecureCRT 断开（右击标签 serial-comx，点击断开连接），打开串口网络调试工具 USR-TCP232-Test，如下图 11.4- 1 所示：

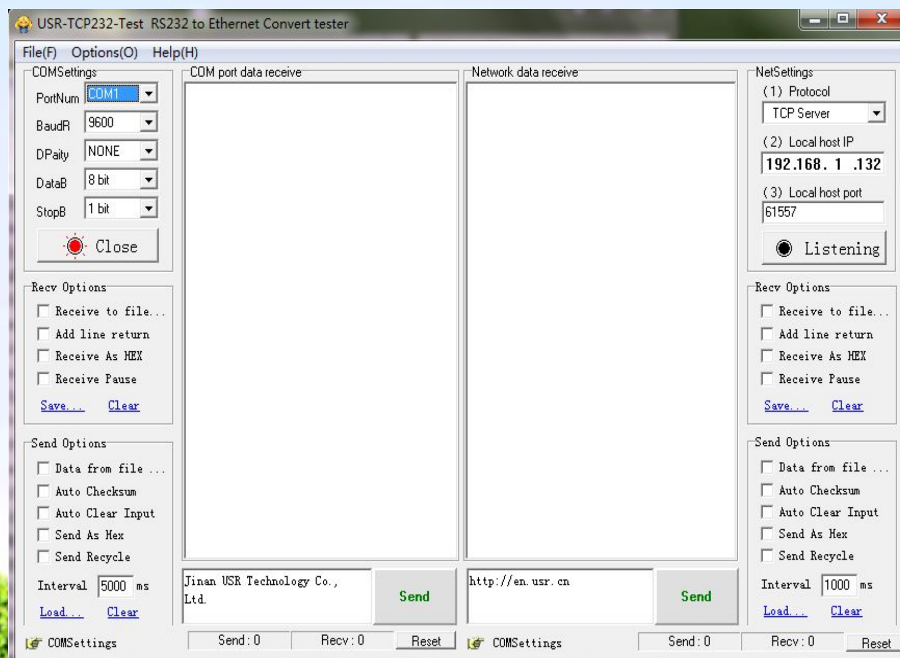


图 11.4- 1 打开串口网络调试工具

左边我们选好 COM 口（该 COM 口和终端所用的是同一个串口），波特率为 57600，打开串口。右侧 protocol 里选择 UDP，Local host IP 填入 PC 机的 IP，一



般都默认给出了，我的是 192.168.1.132，端口号是 61557，这个是应用程序里定义的（详情请看应用程序代码分析）。两侧都不要勾选 Receive As HEX 然后点击 connect，如图 11.4- 2 所示：

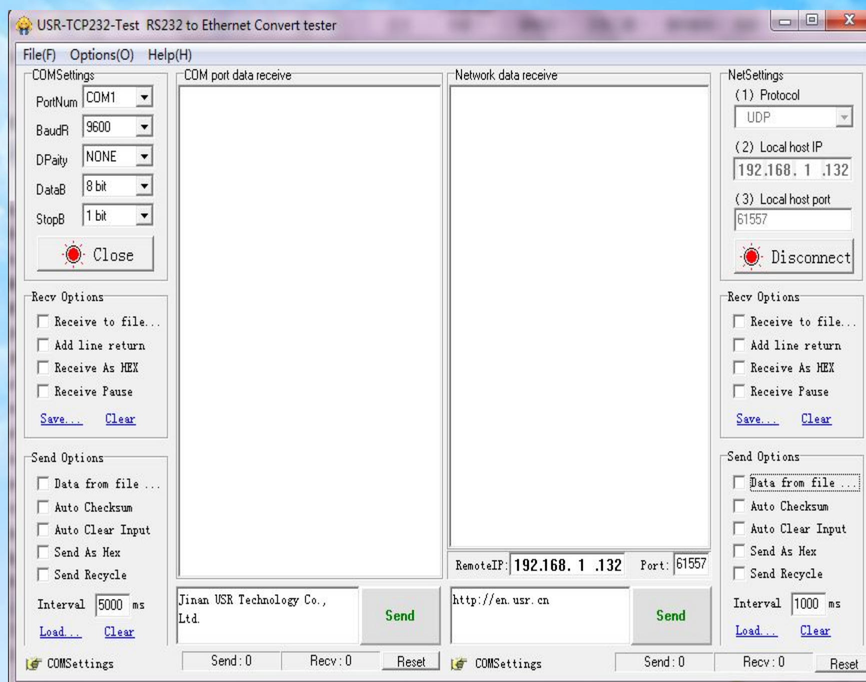


图 11.4- 2 串口调试助手界面

22.4.2 从串口到 UDP

点击左半部分的串口发送按钮“send”如图 11.4- 3 所示：

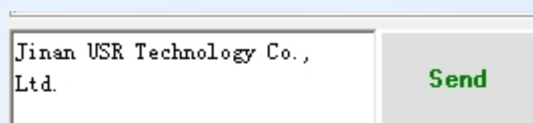


图 11.4- 3 发送窗口

就可以看到右侧的窗口出现数据，如图 11.4- 4 所示：

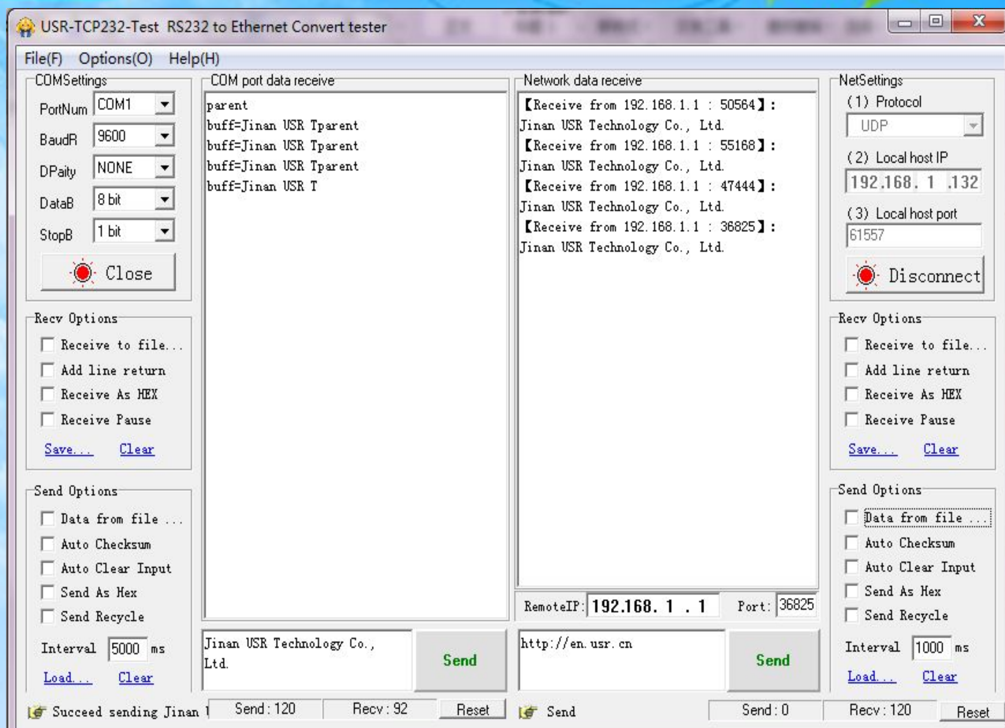


图 11.4- 4 发送和接收示例

这说明，数据通过串口发送到了开发板里，应用程序将其转化为 UDP 广播包，发送了出去，因为 PC 机和开发板在同一网段，当然就接到这些 UDP 数据包啦，同时下面还自动显示了开发板的 ip 和端口号：

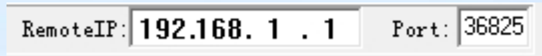


图 11.4- 5 显示开发板的 ip 和端口号

22.4.3 从 UDP 到串口

因为是电脑发送 UDP 包，所以要指定远程 IP 和端口号，所以 RemoteIP 要写成 192.168.1.1（开发板的默认 IP）或者是 192.168.1.255（广播地址），这两个地址开发板都能接收到数据，应用程序监听的端口号是 8003（看程序详解），所以 Port 设置为 8003，然后点击右侧的“Send”，如图 11.4- 6 所示：

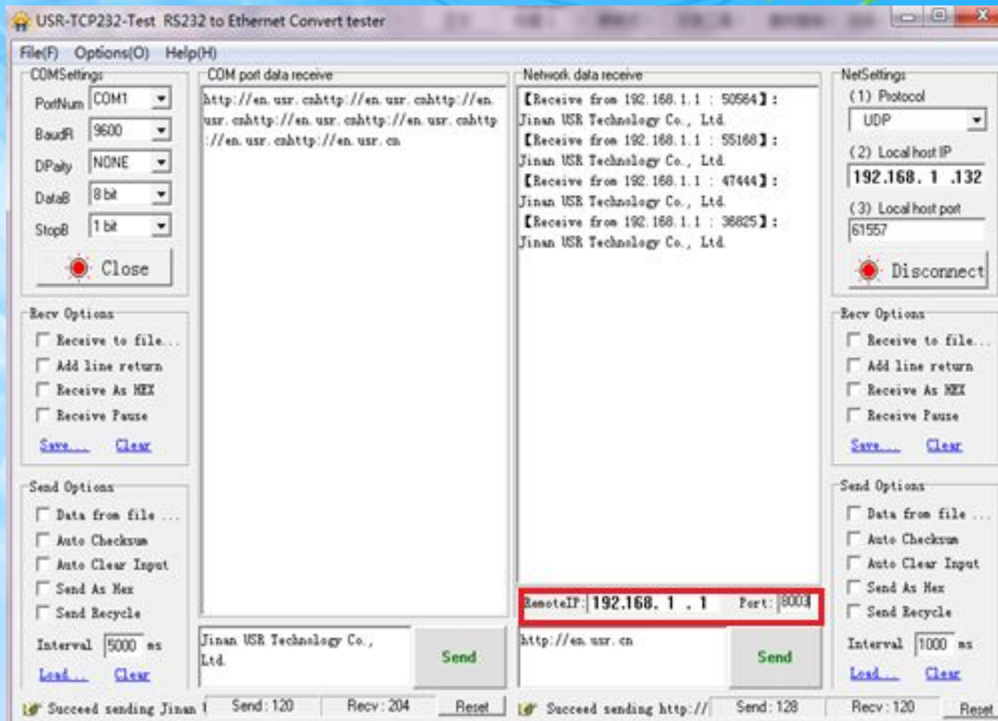


图 11.4- 6 串口数据传输示例

可以看到左侧的窗口有数据出现了。这是 PC 机发出的 UDP 数据包，开发板接收到了，又通过串口发送出来的数据。

至此，这一部分就全部讲解完了！

22.5 项目总结

总体来说，这是一个综合的应用程序，它包括了 Linux 下串口，网络的编程，涉及了文件的读写，系统串口，进程的概念，套接字，UDP 网络等很多知识，具有典型意义，由于开发板是基于 Linux 系统的，设计的 Linux 系统的知识较多，所以需要读者有一定得基础，才能读懂。所以希望读者多多查阅资料，或上网寻找答案，这样才能真正的将它学会。

第 23 章 分布式数据采集与网络自动上传

23.1 概述

该项目包括三部分,一个单片机工程 DHT11.eww，用于采集温湿度，通过串口将数据传到中央模块；一个中央模块下的应用程序 serial_file_ftp.c，编译得到的可执行文件 serial_file_ftp.o（文件的名字可以任意取）；一个脚本文件 lftp.sh。

此应用程序实现的功能：

由单片机采集温度和湿度，通过串口传给中央模块，中央模块的应用程序读

取串口数据（当前的温度与湿度）并存入指定文件中，并以系统时间命名文件。最后利用 shell 脚本编程 lftp.sh 将含有温湿度的文件上传至主机的指定文件夹下。

23.2 相关知识介绍

23.2.1 DHT11 温湿度传感器简介

DHT11 数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器。它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性与卓越的长期稳定性。

1、DHT11 的典型应用电路如图 12.2- 1 所示：

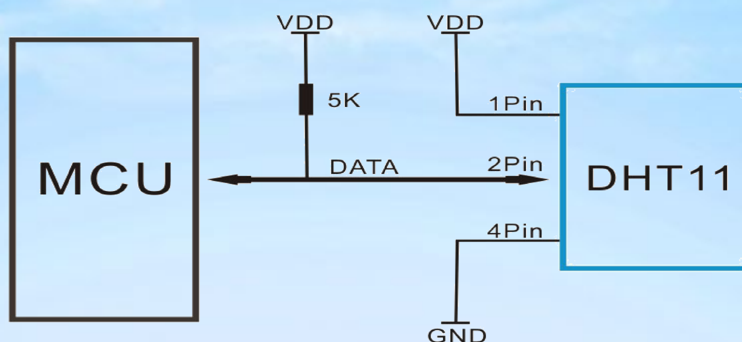


图 12.2- 1DHT11 的典型应用电路

3、DHT11 的引脚功能如表 12.2- 1 所示：

表 12.2- 1DHT11 的引脚功能

Pin	名称	注释
1	VDD	供电 3－5.5VDC
2	DATA	串行数据，单总线
3	NC	空脚，请悬空
4	GND	接地，电源负极

3、DHT11采用的是单线双向的串行接口

DATA 引脚用于微处理器与 DHT11之间的通讯和同步,采用单总线数据格式,一次通讯时间4ms左右,数据分小数部分和整数部分,具体格式在下面说明,当前小数部分用于以后扩展,现读出为零.操作流程如下：

一次完整的数据传输为40bit,高位在前。

数据格式:8bit湿度整数数据+8bit湿度小数数据
+8bi温度整数数据+8bit温度小数数据
+8bit校验和

数据传送正确时校验和数据等于**“8bit湿度整数数据+8bit湿度小数数据+8bit温度整数数据+8bit温度小数数据”**所得结果的末8位。

用户MCU发送一次开始信号后,DHT11从低功耗模式转换到高速模式,等待主机开始信号结束后,DHT11发送响应信号,送出40bit的数据,并触发一次信号采

集,用户可选择读取部分数据.从模式下,DHT11接收到开始信号触发一次温湿度采集,如果没有接收到主机发送开始信号,DHT11不会主动进行温湿度采集.采集数据后转换到低速模式。通讯过程如图12.2- 2所示

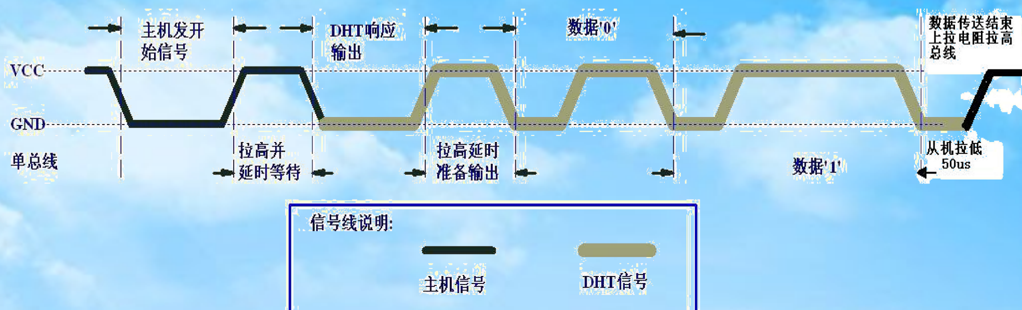


图 12.2- 2DHT11 时序图 1

总线空闲状态为高电平,主机把总线拉低等待DHT11响应,主机把总线拉低必须大于18毫秒,保证DHT11能检测到起始信号。DHT11接收到主机的开始信号后,等待主机开始信号结束,然后发送80us低电平响应信号.主机发送开始信号结束后,延时等待20-40us后, 读取DHT11的响应信号,主机发送开始信号后,可以切换到输入模式,或者输出高电平均可, 总线由上拉电阻拉高。如图12.2- 3所示:

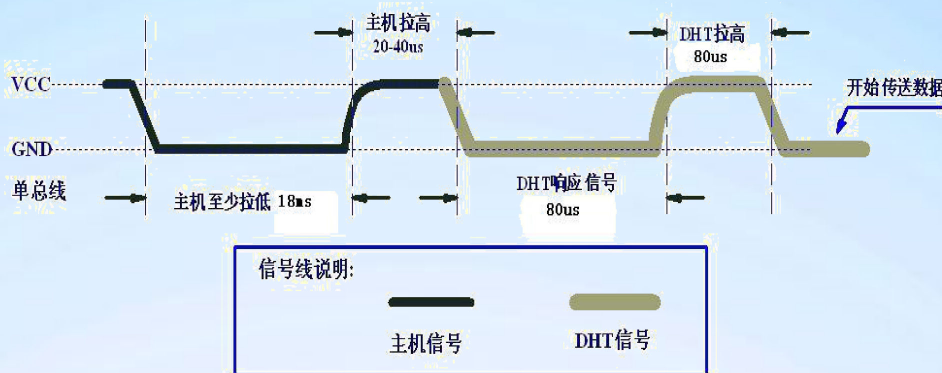


图 12.2- 3DHT11 时序图 2

总线为低电平,说明DHT11发送响应信号,DHT11发送响应信号后,再把总线拉高80us,准备发送数据,每一bit数据都以50us低电平时隙开始,高电平的长短定了数据位是0还是1.格式见下面图示.如果读取响应信号为高电平,则DHT11没有响应,请检查线路是否连接正常.当最后一bit数据传送完毕后, DHT11拉低总线50us,随后总线由上拉电阻拉高进入空闲状态。

数字0信号表示方法如图12.2- 4所示:

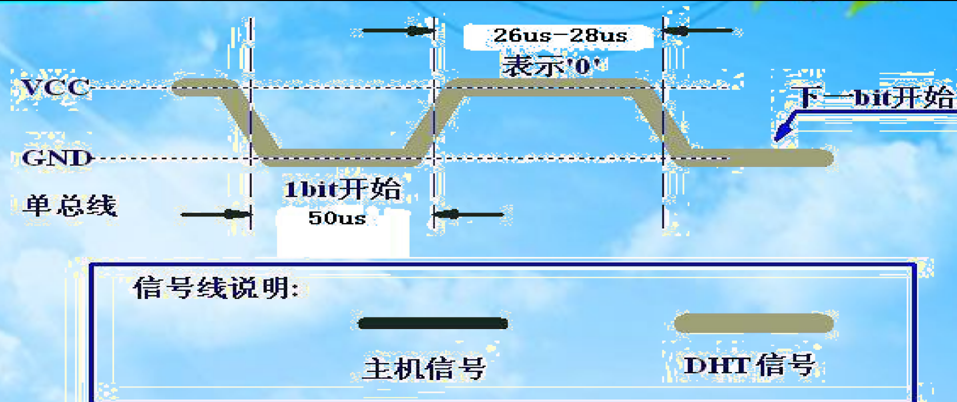


图 12.2- 4DHT11 时序图 3

数字1信号表示方法. 如图12.2- 5所示：

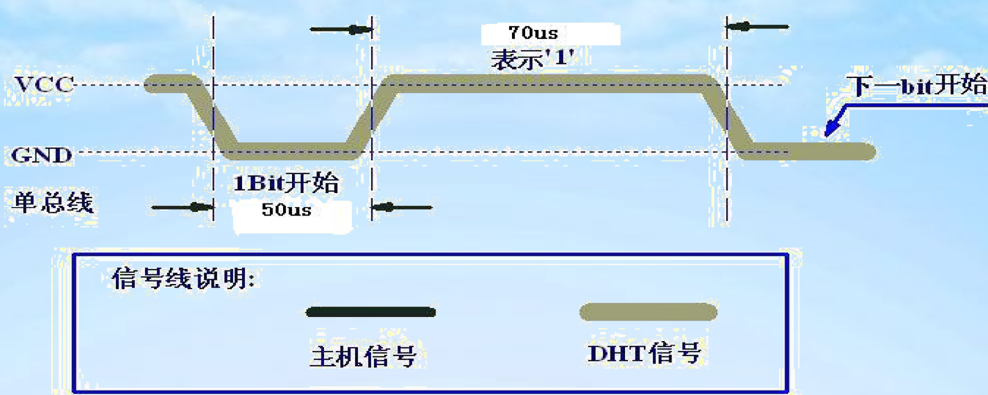


图 12.2- 5DHT11 时序图 4

23.2.2 Linux 下串口的编程简介

参照 STM32+linux 开发板实战篇第十一章的内容

23.2.3 Linux 下文件操作编程简介

文件是 Linux 中的一个重要概念。他们为操作系统和设备提供了一个简单而统一的接口。在 Linux 中，几乎一切都可以看做是文件 。

1）操作

文件操作的总流程是：打开文件-操作文件-关闭文件。

打开文件，就是系统为文件建立缓冲区，并将文件和缓冲的信息写入 FILE 型结构中。使得能够通过缓冲区进行文件的输入和输出操作。

文件使用完后，系统将缓冲区中的数据做相应的处理（如将数据写入文件等），然后，释放缓冲区。这个过程叫做关闭文件。关闭文件的结果，使应用程序不能再对该文件进行输入输出等操作。

2）打开文件和关闭文件

文件的打开和关闭操作是分别由系统函数 fopen()和 fclose()完成的。

打开文件函数 fopen()的格式为：

FILE *fopen(char*filename,char*mode)

其中参数 filename 是字符型指针，它指向的字符串是要打开的文件名。参数 mode 也是字符型指针，它指向的字符串是文件的使用方式，称为打开模式。

这两个参数的实参可以是字符串常量，可以是指向字符串的指针，也可以是字符数组的首地址。

文件的使用方式字符串共有 12 个，其中 6 个是用于文本文件的，6 个用于二进制文件的。

参数说明：

*filename：打开文件的文件名

*mode：打开的方式

r 以只读方式打开文件，该文件必须存在。

r+ 以可读写方式打开文件，该文件必须存在。

rb+ 读写打开一个二进制文件，允许读数据。

rw+ 读写打开一个文本文件，允许读和写。

w 打开只写文件，若文件存在则文件长度清为 0，即该文件内容会消失。若文件不存在则建立该文件

w+ 打开可读写文件，若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件。

fopen 在成功时返回一个非空的 FILE *指针。失败返回 NULL

关闭文件函数 fclose()的格式是：

int fclose(FILE*fp)

fclose 函数关闭指定的文件流 stream，此操作会使所有未写出的数据写出。因为 stdio 库函数会对数据进行缓冲，所以调用 fclose 函数很重要。如果程序需要确保数据已经全部写出，就应该调用 fclose 函数。

当正确关闭指定的文件时，函数返回 0；否则返回非 0。

3）函数 fprintf()

格式化写函数，其格式为：

int fprintf(FILE*fp,char*control_string,e1,e2,...,en)

其中参数 fp 是文件型指针；control_string 是存放格式字符串的字符常量，或者是存放格式字符串的数组首地址，或者是指向格式字符串的指针变量；e1,e2,...,en 是与格式字符串匹配的变量地址或数组首地址。

函数的功能是，从 fp 指向的文件中，按照 control_string 规定的格式，读取 n

个数据，依次存入 e1,e2,...,en 地址中。如果读操作正确，则返回所读数据的数目；错误，则返回 EOF（-1）。

当读取数据后，内部指针自动移到下一个未读数据的位置。

23.2.4 日期时间函数

time()取得目前的时间

函数原型：time_t time(time_t*t);

函数返回值：成功则返回秒数，失败则返回（-1）值。

函数说明：此函数会返回公元 1970 年 1 月 1 日的 UTC 时间从 0 时 0 分 0 秒起到现在所经过的秒数。如果 t 为空指针的话，此函数也会将返回值存到 t 指针所指的内存。

ctime()将时间和日期以字符串格式表示

函数原型：char *ctime(const time_t *timep);

函数说明：ctime（）将参数 timep 所指的 time_t 结构中的信息转换成真实世界所使用的时间日期表示方法，然后将结果以字符串形式返回。

函数返回值：返回一个字符串表示目前当地的时间日期。这个字符串的长度是固定的，为 26。

取得当前日期和时间：

```
#include      <stdio.h>
#include      <time.h>
main（）
{
    time_t timep;
    time(&timep);
    printf("%s",ctime(&timep));
}
```

strftime()函数表示格式化时间

头文件：time.h

函数原型：我们可以使用 strftime()函数将时间格式化为我们想要的格式。它的原型如下：

size_t strftime(char* strDest,size_tmaxsize,const char *format,const struct tm *timeptr)

参数说明：

我们可以根据 format 指向字符串中格式命令把 timeptr 中保存的时间信息放在 strDest 指向的字符串中，最多向 strDest 中存放 maxsize 个字符。该函数返回向 strDest 指向的字符串中放置的字符数。

函数 strftime()的操作有些类似于 sprintf()：识别以百分号(%)开始的格式命令集合，格式化输出结果放在一个字符串中。格式化命令说明串 strDest 中各种日期和时间信息的确切表示方法。格式串中的其他字符原样放进串中。格式命令列在下面，它们是区分大小写的。

%a 星期几的简写

%A 星期几的全称
%b 月份的简写
%B 月份的全称
%c 标准的日期的时间串
%C 年份的后两位数字
%d 十进制表示的每月的第几天
%D 月/天/年
%e 在两字符域中，十进制表示的每月的第几天
%F 年-月-日
%g 年份的后两位数字，使用基于周的年
%G 年份，使用基于周的年
%h 简写的月份名
%H 24 小时制的小时
%I 12 小时制的小时
%j 十进制表示的每年的第几天
%m 十进制表示的月份
%M 十时制表示的分钟数
%n 新行符
%p 本地的 AM 或 PM 的等价显示
%r 12 小时的时间
%R 显示小时和分钟：hh:mm
%S 十进制的秒数
%t 水平制表符
%T 显示时分秒：hh:mm:ss
%u 每周的第几天，星期一为第一天 （值从 1 到 7，星期一为 1）
%U 第年的第几周，把星期日作为第一天（值从 0 到 53）
%V 每年的第几周，使用基于周的年
%w 十进制表示的星期几（值从 0 到 6，星期天为 0）
%W 每年的第几周，把星期一做为第一天（值从 0 到 53）
%x 标准的日期串
%X 标准的时间串
%y 不带世纪的十进制年份（值从 0 到 99）
%Y 带世纪部分的十制年份
%z，%Z 时区名称，如果不能得到时区名称则返回空字符。
%% 百分号

23.3 应用程序代码分析

23.3.1 单片机工程

学习了 STM32 之后，相信对于如何进行单片机编程已经不再陌生，这里不



再赘述。开发板左侧有 SW 的下载接口，可以下载程序。温湿度传感器的采集实际上是一线总线，把握好时间的控制，是不难读取温湿度，这里只看一下主函数，以及代码实现的主要功能，详细代码请看参看源代码。

```
#include "stm32f0xx.h"
#include "DHT11.h"
#include "uart2.h"
unsigned char table[5]={0,0,0,0};
int main(void)
{
  Delay(5000);
  Delay(5000);
  Delay(5000);
  DHT11_GPIO_Init();
  USART_Configuration();
  while(1)
  {
    r_RH_T(table);
    UART_send_byte(table[0]);
    UART_send_byte(table[1]);
    UART_send_byte(table[2]);
    UART_send_byte(table[3]);
    Delay(50000);
  }
}
```

`r_RH_T();` 读取温湿度的函数；主函数先是有一段延时，然后对温湿度传感器进行初始化，对串口进行配置，在主循环里不断的读取温湿度的数据存放到 table 数组里，然后再通过串口发送出去。

23.3.2　中央模块下的应用程序

首先是头文件，应用程序需要包含以下头文件：

```
#include     <stdio.h>       /*标准输入输出定义*/
#include     <stdlib.h>      /*标准函数库定义*/
#include     <unistd.h>
#include     <sys/types.h>   //数据类型
#include     <sys/stat.h>    /*定义了一些返回值的结构。*/
#include     <fcntl.h>       /*文件控制定义*/
#include     <termios.h>     /*PPSIX 终端控制定义*/
#include     <errno.h>
#include     <string.h>
#include     <time.h>
#define FALSE  -1
#define TRUE   0
```

图 12.3- 1 所包含的头文件

然后是波特率和校验位的配置，以读写方式打开串口，IO 口为非阻塞模式。详细代码见 Linux 下串口与网络的编程。

图 12.3- 2 是主函数的代码：




```
int main(int argc, char **argv){
    int fd,i=0,sum=0;
    int nread,nwrite;
    unsigned char buff[10];
        unsigned char readbuff[10];
    unsigned char filebuff[50]={"1234"};
    char *dev  = "/dev/ttyS0";
    //获取当前时间的函数
    time_t nowtime;
    struct tm *timeinfo;
    char txtname[100];
    fd = OpenDev(dev);
    set_speed(fd,115200);
    if (set_Parity(fd,8,1,'N') == FALSE)  {
        printf("Set Parity Error\n");
        exit (0);
    }

    while (1)
    {
        tcflush(fd, TCIOFLUSH);   //刷新串口
        bzero(readbuff,100);      //清空缓冲区
        bzero(buff,100);
        while((nread = read(fd, buff,1))<=0)
        sum=1;
        nread=0;
        usleep(500000);                          //要让串口有足够的时间接数
        while((nread = read(fd, readbuff,8))>0)  //接收串口的数据
        {

        for(i=0;i<nread;i++)
         {
            buff[sum+i]=readbuff[i];
         }
        sum=nread+sum;
        nread=0;
        bzero(readbuff,100);
        usleep(10000);
        }
```

图 12.3- 2 主函数 1

```
        printf("buff = %s",buff);
        tcflush(fd, TCIOFLUSH);          //丢弃要写入的对象
        sum=0;
        time(&nowtime);
        timeinfo=localtime(&nowtime);    //获取当前时间
        strftime(txtname,100,"DHTdata_%Y%m%d_%H%M%S.txt",timeinfo);
        FILE *fp;
        if((fp=fopen(txtname,"w"))==NULL)//打开文件 没有就创建
        {
            printf("can not creat file!\n");
        }

        fprintf(fp,"the humidity is %c%c,the temperature is %c%c",
        buff[0],buff[1],buff[2],buff[3]);
        //传送格式化输出到文件中
        fclose(fp);
        bzero(buff,100);
        system("sh lftp.sh");
    }
    //close(fd);
    // exit (0);
}
```

图 12.3- 3 主函数 2

在主函数中，首先打印出一段提示语，即拔掉串口线，跳上跳帽，目的是断开终端串口，然后使串口与 STM32 单片机相连接，其硬件原理可参考第一部分第二章硬件原理部分内容；其次是打开串口，等待串口接收数据，数据存到 buff 里，然后是 time()函数，获取当前时间，将时间转换成字符串，存到 txtname 里，以 txtname 的名字创建文件，然后向文件里写入字符串“the humidity is %c%c,the temperature is %c%c”，%c 的位置是接收到的字符串，也就是温度和湿度的数据。strftime(txtname,100,"DHTdata_%Y%m%d_%H%M%S.txt",timeinfo); 表示格式化当前时间，以当前系统时间命名文件夹。

最后用 system（）函数调用 lftp.sh 脚本文件。

23.3.3 shell 脚本文件

system("sh lftp.sh");表示执行脚本文件 lftp.sh，其功能是将已经创建的以系统时间命名的文件上传至 PC 机，脚本文件内容如图 12.3- 4 所示：

lftp - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#!/bin/bash
chmod 777 DHTdata*
lftp gxt:1@192.168.1.123 <<!
mput DHTdata*
quit
!
rm DHTdata*
```

图 12.3- 4 lftp.sh

第一行修改创建文件的权限，第二行 lftp 拨号，用户名是 gxt，密码是 1，通过此命令行就建立了开发板与上位机的 lftp 连接。第三行 mput /www/DHTdata*实现的功能是将/www 目录下的所有以 DHTdata 作为文件名开头的文件批量上传到上位机。上位机的 IP 为 192.168.1.123，在执行此程序之前，上位机要设置 lftp 的服务端，用户名和密码都要配置好。

23.4 应用程序执行

应用程序功能实现过程中，需要用到 lftp 上传，所以要保证上位机与开发板保持正常的网络通信（有线，无线或3G）并且上位机要打开 FileZilla server 来开启 ftp 服务器。在脚本里我们进行 lftp 拨号时用的上位机的 IP 为192.168.1.123，因此最好将上位机的 IP 修改为192.168.1.123(也可以在命令提示符下用 ipconfig 命令查看 IP，同时也要修改 lftp.sh 中的 IP)。

23.4.1 应用程序的运行结果

运行前，先对 FileZilla Server 配置（参看预备篇第一章 1.6 节）。

我们将采集温湿度的程序下载到 STM32 单片机里，然后将 serial_file_ftp.c 交叉编译出的可执行文件 serial_file_ftp.o，用 WinSCP 将可执行文件 serial_file_ftp.o 与脚本文件 lftp.sh 上传到开发板的 Program 文件夹下，修改文件的权限，在终端可以看到如图 12.4- 1 所示：

```
root@MicroCloud:/Program# ls
helloworld.o        ser2web             seril_udp
lftp.sh             serial_file_ftp.o   udp_serial.o
root@MicroCloud:/Program#
```

图 12.4- 1 lftp.sh 和 serial_file_ftp.o 文件

可以看到 serial_file_ftp.o 和 lftp.sh 文件，输入执行命令：

```
root@MicroCloud:/Program# ./serial_file_ftp.o
```

当显示上图内容时，表示程序已经运行，在这之前要调好跳线帽，如图 12.4- 2 所示

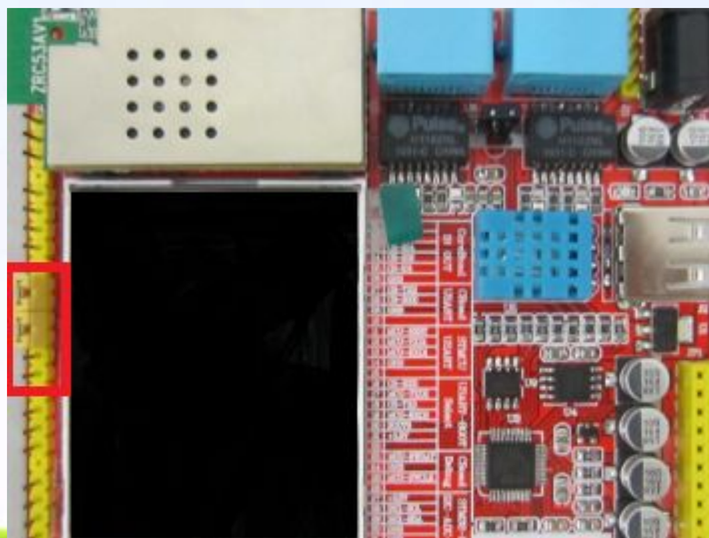


图 12.4- 2 跳帽切换

带有 DHT 字样的文件就是已上传到电脑上的文件，这里上位机存放文件的路径是 E：\ftpserver,打开可以看到如图 12.4- 3所示内容：

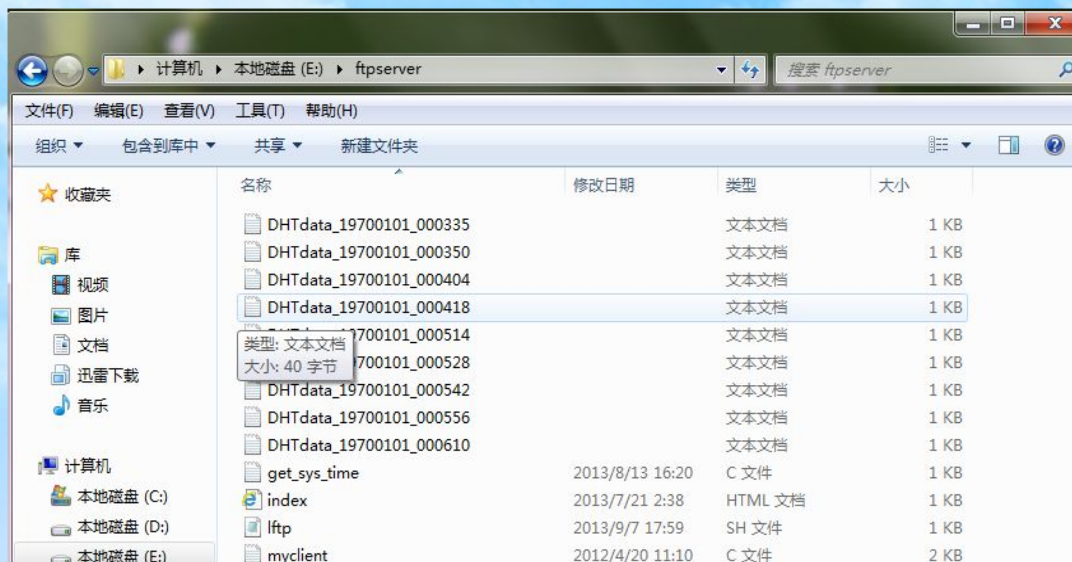


图 12.4- 3 电脑上的文件

随意打开一个带 DHTdata 字样的文件，内容如图 12.4- 4：



图 12.4- 4 DHTdata 文件内容

上述文件中清晰的记录了某一时刻的温湿度。

23.5 项目总结

与上一章相比，本章的内容比较容易弄懂，但涉及的文件类型较多，有单片机程序，C 语言程序和脚本程序，调试时可能不会顺利的一次调通，可借助其他工具进行分块调试，比如 lftp 部分可先检测 lftp 是否能成功上传文件（开发板功能篇 第五章），然后再整体运行程序，只有认真的保证每一步都配置正确，才能最终运行成功。

第 24 章 监控拍照与远程网络自动传输

24.1 概述

该项目只包含一个文件 video_lftp.sh，集合了使能和开启摄像头，截取图像，进行 ftp 连接和上传文件等所有的命令。

24.2 应用程序代码分析

图 13.2- 1 所示是 video_lftp.sh 脚本文件的内容：

```
#! /bin/bash
/etc/init.d/mjpg-streamer start
mjpg_streamer -i "input_uvc.so -r 352x288 -f 15 -q 80 -y"  -o "output_http.so -p 8080 -w /www"&
sleep 2
wget "http://localhost:8080/?action=snapshot" -O /tmp/video_$(date +"%Y%m%d_%H%M%S").jpg
killall mjpg_streamer
chmod 777 /tmp/video*
lftp gxt:1@192.168.1.123 <<!
put /tmp/video*
quit
!
rm s.jpg
```

图 13.2- 1 video_lftp.sh 脚本文件

第一行/etc/init.d/mjpg-streamer start， 作用是使能 mjpg-streamer。mjpg-streamer 是一款摄像头驱动软件，mjpg_streamer -i "input_uvc.so -r 352x288 -f 15 -q 80 -y" -o "output_http.so -p 8080 -w /www" 表示启动摄像头，若在终端执行这两个命令，在火狐或谷歌浏览器地址栏中输入 http://192.168.1.1:8080/?action=stream 便可以看到摄像头监视的画面。sleep 2 延时两秒，目的是等待摄像头启动完全。wget”http://localhost:8080/?action=snapshot”-o/tmp/video_$(data+”%Y%m%d_%H%M%S”).jpg 表示从网页上获取图片，存放到/tmp/里并以系统时间命名。killall mjpg_streamer：关闭摄像头的进程。接下来的是修改文件权限，和 lftp 上传文件，之前都已涉及过，这里不再赘述。

24.3 应用程序执行

首先确保 PC 机和开发板网络是连接好的（有线或者无线参看功能篇第 4 章 4.2 节）,由于脚本文件 video_lftp.sh 里的 IP 是 192.168.1.123，所以要将上位机的 IP 修改为 192.168.1.123（也可以修改 video_lftp.sh 文件里的 IP 和上位机一致）然后打开 FileZilla Server 软件（参看开发板预备篇第一章 1.6 节的内容），如 图 13.3- 1 所示：



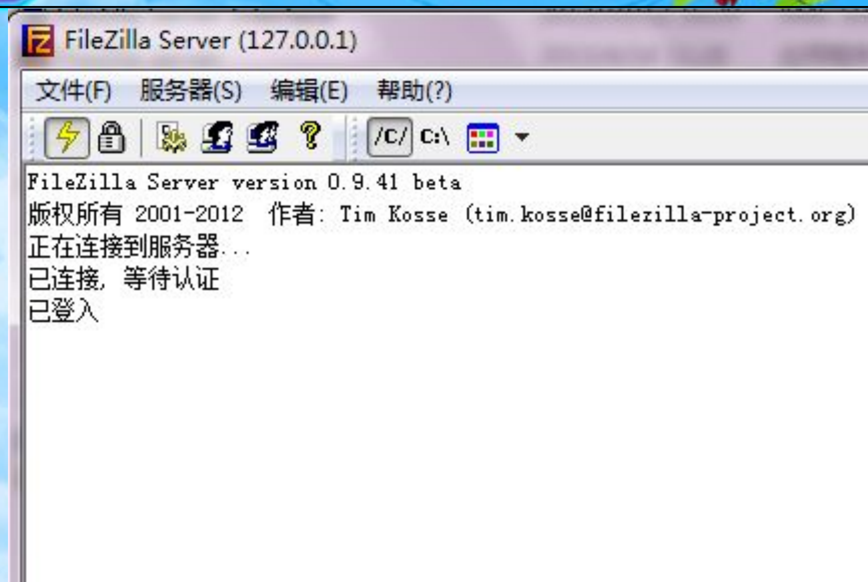


图 13.3- 1FileZilla Server 软件

将摄像头插入开发板的扩展 USB 口（这个时候需要一个 HUB 来同时插入 u 盘和摄像头），给开发板上电，将脚本文件 video_lftp.sh 用 WinSCP 上传至开发板的 Program 目录下，打开 SecureCRT，可以看到/dev 目录下出现 video0(代表刚刚插入的摄像头设备)，如图 13.3- 2 所示：

```
cd /dev
root@MicroCloud:/dev# ls
audio            mtd0ro         mtdblock3            shm
autofs           mtd1           mtdblock4            snd
bus              mtd1ro         mtdblock5            tty
console          mtd2           network_latency      ttyS0
cpu_dma_latency  mtd2ro         network_throughput   ttyS1
dsp              mtd3           null                 ttyUSB0
etherd           mtd3ro         ppp                  ttyUSB1
full             mtd4           ptmx                 urandom
input            mtd4ro         pts                  video0
kmsg             mtd5           random               watchdog
log              mtd5ro         sda                  watchdog0
mem              mtdblock0      sdb                  zero
mixer            mtdblock1      sg0
mtd0             mtdblock2      sg1
```

图 13.3- 2 /dev 文件夹下内容

在终端输入命令：sh /Program/video_lftp.sh 执行程序:

```
root@MicroCloud:/tmp# sh /Program/video_lftp.sh
MJPG Streamer Version: svn rev: exported
 i: Using V4L2 device.: /dev/video0
 i: Desired Resolution: 352 x 288
 i: Frames Per Second.: 15
 i: Format............: YUV
 i: JPEG Quality......: 80
Adding control for Pan (relative)
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
Adding control for Tilt (relative)
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
Adding control for Pan Reset
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
Adding control for Tilt Reset
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
Adding control for Pan/tilt Reset
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
Adding control for Focus (absolute)
UVCIOC_CTRL_ADD - Error: Inappropriate ioctl for device
mapping control for Pan (relative)
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Tilt (relative)
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Pan Reset
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Tilt Reset
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Pan/tilt Reset
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Focus (absolute)
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for LED1 Mode
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for LED1 Frequency
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Disable video processing
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
mapping control for Raw bits per pixel
UVCIOC_CTRL_MAP - Error: Inappropriate ioctl for device
 o: www-folder-path...: /www/
 o: HTTP TCP port.....: 8080
 o: username:password.: disabled
 o: commands..........: enabled
Connecting to localhost:8080 (127.0.0.1:8080)
video_20131014_12223 100% |*******************************| 16747   0:00:00 ETA
root@MicroCloud:/tmp#
```

图 13.3- 3 程序启动信息

出现上图的信息说明摄像头图片上传成功，此时可以看到 PC 机共享文件夹里出现了刚刚上传的图片，并且是以系统时间命名的，如 图 13.3- 4 所示：

名称	修改日期	类型	大小
get_sys_time	2013/8/13 16:20	C 文件	1 KB
index	2013/7/21 2:38	HTML 文档	1 KB
lftp	2013/9/7 17:59	SH 文件	1 KB
myclient	2012/4/20 11:10	C 文件	2 KB
video_20131014_122002	2013/10/14 20:20	JPEG 图像	18 KB
video_20131014_122148	2013/10/14 20:21	JPEG 图像	16 KB
video_20131014_122239	2013/10/14 20:22	JPEG 图像	17 KB
微云电子出品	2013/8/23 19:36	WinRAR 压缩文件	467,518 KB

图 13.3- 4 上传的图片



24.4 项目总结

本章项目实现功能比较简单，主要是开启摄像头拍摄图片以及 lftp 上传文件，这些在前面都有所涉及，这里只是将这两部分功能相结合，因此叙述比较简略，若有不理解之处请参看 STM32+LINUX 开发板功能篇。

第 25 章 远程短信监控与网络邮件事件互动

25.1 概述

本章项目实现的主要功能是：利用开发板给邮箱发送邮件，邮箱有短信提醒功能，会把收到的邮件以短信的形式发送给绑定手机，通过手机回复邮件，开发板来提取回复邮件正文，将邮件正文作为命令来控制 led 亮灭。在功能篇中已经介绍过了开发板收发邮件的过程，这里不再赘述。

该项目包括两部分，一个单片机工程 message.uvproj，负责接收串口 ttyS0 发送的控制命令，从而采取相应的操作；recv_message.sh，负责收发邮件及邮件正文提取。

项目总体功能：开发板发送邮件到指定邮箱，邮箱进行手机短信提醒，手机回复邮件，开发板提取邮件正文控制相应外设，例如 led 亮灭。

25.2 相关知识介绍

我们除了需要准备一部具有短信收发功能的手机外，还需要注册一个 139 邮箱，因为 139 邮箱较其他邮箱的主要优势在于：

（1）可以以客户手机号作为邮箱的客户名，他人发送邮件更加方便；

（2）139 邮箱提供邮件到达短信提醒功能；

（3）客户可以通过 139 邮箱发送彩信至客户手机上。

至于收发邮件方面的相关知识，详情参见 STM32+LINUX 开发板功能篇。

25.3 应用程序代码分析

25.3.1 recv_message.sh 脚本分析

此脚本的功能是为开发板创建一个 root 文件，来存放收到的邮件。并给 root 文件赋予可执行权限。过程已经在 STM32+LINUX 开发板功能篇收发邮件章节详细介绍过了，这里不再赘述。

（2）下面分析脚本：recv_message.sh

```
fetchmail -v -f /etc/fetchmailrc -P 110
cat /tmp/spool/mail/root | grep -A 5 "Subject: Re:test" > /etc/cmdline
awk 'NR==6{print}' /etc/cmdline > /dev/ttyS0
rm /tmp/spool/mail/root
touch /tmp/spool/mail/root
chmod 777 /tmp/spool/mail/root
```

图 14.3- 1recv_message.sh 脚本

第一行 fetchmail -v -f /etc/fetchmailrc -P 110。-v：显示输出调试信息，会把我们接收的信息显示出来，-f：指定运行控制文件，即按照 fetchmailrc 文件里配置的去接收邮件,-P：指定端口号为 110。

第二行 cat /tmp/spool/mail/root 命令意思是查看我们创建的 root 文件，前面已经说过了 fetchmail 会把收到的邮件自动放在 root 里。

第三行 cat /tmp/spool/mail/root | grep -A 5 “Subject: Re:test” > /etc/cmdline。cat /tmp/spool/mail/root 是查看我们的邮箱，grep 是提取命令，-A 5：显示匹配行之后的第 5 行，因为匹配行之后的第五行就是正文内容。

```
From root  Mon Sep  2 07:41:13 2013
X-Richmail-Antispam: sCL2rVi0borhSzeOjpYxWEmrxqxSIjpywIkDhRS3bNKNy5SIjpgwYtQ=
X-RM-SPAM-FLAG:00000000
Received: from 121.195.184.79 [121.195.184.79]
        by 127.0.0.1 with POP3 (fetchmail-6.3.26)
        for <root@localhost> (single-drop); Mon, 02 Sep 2013 07:41:13 +0000 (GMT)
Received:from appfetion4 (unknown[172.16.72.141])
        by rmsmtp-cmudms-5-15-12020 (RichMail) with SMTP id 2ef45223f14ebb1-1bbb4;
        Mon, 02 Sep 2013 10:00:46 +0800 (CST)
X-RM-TRANSID:2ef45223f14ebb1-1bbb4
Date: Mon, 2 Sep 2013 10:00:46 +0800 (CST)
From: 15146843614@139.com
To: 15146843614@139.com
Message-ID: <1288814725.1409897137808724649 6.JavaMail.moSMS@appfetion4>
Subject: Re:test
MIME-Version: 1.0
Content-Type: text/plain;charset=GBK
Content-Transfer-Encoding: quoted-printable

1
```

图 14.3- 2 查看邮箱存储文件

“Subject：Re：test”是匹配行，也就是收到邮件的主题。因为我们发送时用的主题是 test，回复时就是 Re：test。最后的>/etc/cmdline 的意思是把提取出来的内容存在 etc/cmdline 里。

第四行 awk ‘NR==6{print}’ /etc/cmdline > /dev/ttyS0 此命令的意思是提取/etc/cmdline 文件中的第六行即短信的正文发送到/dev/ttyS0 设备里。为什么要提取第六行呢，查看 cmdline 就会发现，邮件的正文存放在第六行。因此我们提取第六行。发送到 ttyS0 串口能够直接为单片机所用。

第五行 rm /tmp/spool/mail/root 是删除邮件文件，每次接收到邮件后，我们删除它再重新创建，这样收到的邮件就会是最新发送的邮件。touch /tmp/spool/mail/root 就是重新创建 root 命令。

最后 chmod 777 /tmp/spool/mail/root 是给 root 文件赋予执行权限。

25.3.2 单片机软件设计分析

本单元的单片机部分的程序很简单，就是接收串口发送过来的字符，并根据字符来判断，如果是字符“1”，则点亮 led 灯，如果是字符“2”则关闭 led 灯。串口的波特率要设置为 57600。

其主函数的内容如下：

```
int main(void)
{
  Delay(5000);
  Delay(5000);
  Delay(5000);
  LED_Init();
  USART_Configuration();
  while(1)
  {
    ;
  }
}
```

中断函数的内容如下：

```
void USART2_IRQHandler(void)
{
    uint8_t temp;
   if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
   {
    temp = USART2->RDR;   // 从RXFIFO中读取接收到的数据
    if(temp=='1')
      LED_Open();
    else if(temp=='2')
      LED_Close();

   }
   return;
}
```

内容很简单，不细说。

25.4 应用程序执行

首先创建一个在目录/tmp/spool/mail 下的 root 文件（参看功能篇中的接收邮件部分）。

打开 SecureCRT，在终端中输入命令：

mkdir /tmp/spool

mkdir /tmp/spool/mail



touch /tmp/spool/mail/root
chmod 777 /tmp/spool/mail/root

接下来发送邮件：在 SecureCRT 中，输入 echo “发送 1led 亮，发送 2led 灭” | mutt -s “test” *********@139.com，邮箱会以短信形式发送给手机。

手机收到短信之后回复控制命令：回 1，开发板 led 亮；回 2，开发板 led 灭。图 14.4- 1 是邮件的正文：



图 14.4- 1 邮件正文

回复短信之后，执行 recv_message.sh 脚本：./recv_message.sh，此时就会看到 led 的亮灭变化了。每回复一次邮件，就要执行一次 recv_message.sh。

开发板现象：

执行 recv_message.sh 之前开发板现象：



图 14.4- 2 led 点亮

执行 recv_message.sh 之后现象：



图 14.4- 3led 熄灭

25.5 项目总结

这章内容，总体来说有点繁琐，需要注意的事项有很多，需要去多看多理解。最好时常通过电脑登陆 139 邮箱删除收到的邮件，以防开发板提取短信正文的时候提取错误。

要掌握 recv_message.sh 脚本里的每行代码的意思，真正了解本项目的执行流程以及原理。

第 26 章 Web 网页模式之室内环境监控

26.1 概述

该项目包括三部分，一个单片机工程 RR.eww，负责接收 CPU 通过串口发送的命令，从而采取相应的操作；一个网页文件 car.htm；一个 lua 语言编写的简单脚本 web2ser，负责串口和网页之间的通信。

此应用程序实现的功能：通过网页控制下位机的的外设和查看摄像头拍摄的画面。脚本文件用于打开相应的串口并将网页发送下来的命令通过串口发送出去。单片机接收串口命令，根据不同的命令采取相应操作，比如切换图片，点亮 led 等。



26.2 相关知识介绍

26.2.1 Lua 语言

Lua 是一个小巧的脚本语言，其设计目的是为了嵌入应用程序中，从而为应用程序提供灵活的扩展和定制功能。Lua 由标准 C 编写而成，几乎在所有操作系统和平台上都可以编译，运行。Lua 并没有提供强大的库，这是由它的定位决定的。所以 Lua 不适合作为开发独立应用程序的语言，但它可以配合其他的应用程序来实现某些应用程序不容易实现的功能。同一个功能，C 语言可能需要几十行的代码，而 lua 却仅仅只需要几行就可以达到目的。本项目就是用 lua 脚本，来实现系统串口和网页之间数据的传输的。

26.2.2 LCD 液晶彩屏

LCD 液晶彩屏是一种采用液晶控制透光度技术实现色彩的显示器。STM32+LINUX 开发板中的 LCD 是 2.4 寸 TFT LCD 由 320×240 个像素点组成显像。

1）LCD 引脚

表 15.2- 1 中介绍了 STM32+LINUX 开发板中的 LCD 液晶彩屏各个引脚的功能。

表 15.2- 1LCD 液晶彩屏引脚

引脚号	引脚名称	功能	引脚号	引脚名称	功能
1	GND	地	16	DB15	8 位数据总线
2	VDD	电源	17	GND	地
3	IOVCC	电源	18	YD	触控面板
4	CS	片选	19	XL	触控面板
5	RS	选择寄存器	20	YU	触控面板
6	WR	写使能	21	XR	触控面板
7	RD	读使能	22	LED-1	LED 阴极
8	REST	复位	23	LED-2	LED 阴极
9	DB8	8 位数据总线	24	LED-3	LED 阴极
10	DB9	8 位数据总线	25	LED-4	LED 阴极
11	DB10	8 位数据总线	26	LED-5	LED 阴极
12	DB11	8 位数据总线	27	NC	
13	DB12	8 位数据总线	28	LEDA	LED 阳极



14	DB13	8 位数据总线	29	GND	地
15	DB14	8 位数据总线			

2）LCD 液晶彩屏执行命令

LCD 液晶彩屏读数据是首先将片选 CS 拉低，当 RS 拉低时进行命令的输入，当 RS 置高时可以进行数据的写入。将 RS 置高后，通过 RD 的上升沿将数据写入，每次可以写入两个字节，进而可以执行对 LCD 液晶彩屏的操作。

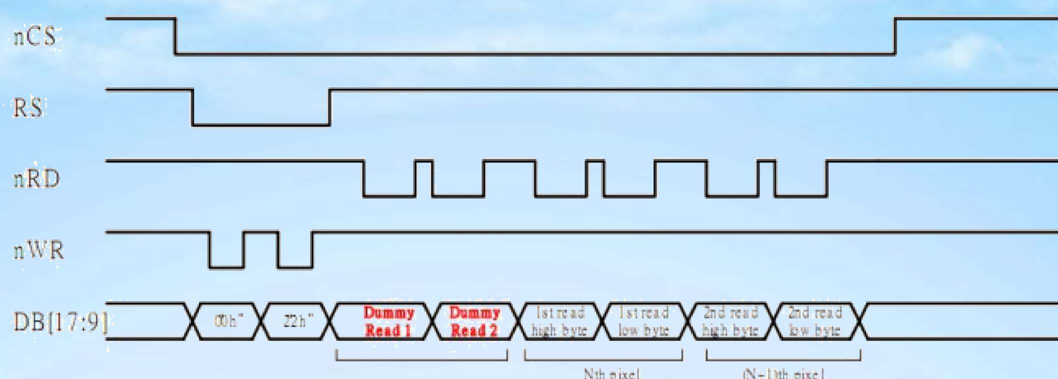


图 15.2- 1LCD 液晶彩屏执行命令时序

LCD 液晶彩屏的初始化配置是已经配置好的，所以只要将每个像素点所需要的两个16进制字节输入到 LCD 液晶彩屏中，就能呈现整个屏幕的图片。

26.2.3 Flash 的读写

Flash 存储芯片可以为用户提供存储解决方案，适合代码下载应用，例如存储声音、文本和数据。工作电压在 2.7V 至 3.6V 之间。W25Q32FVSIG Flash 存储芯片有 16384 可编程页，每页 256 个字节，还具有 1024 个可擦除“扇区”或 64 个可擦除“块”。“页编程指令”每次编程 256 个字节，“扇区擦除指令”每次擦除 16 页，“块擦除指令”每次擦除 256 页，“整片擦除指令”每次擦除整个芯片。

1）Flash 封装及引脚配置

根据图15.2- 2中的 Flash 存储芯片封装，表15.2- 2介绍了 Flash 存储芯片各个引脚的功能。

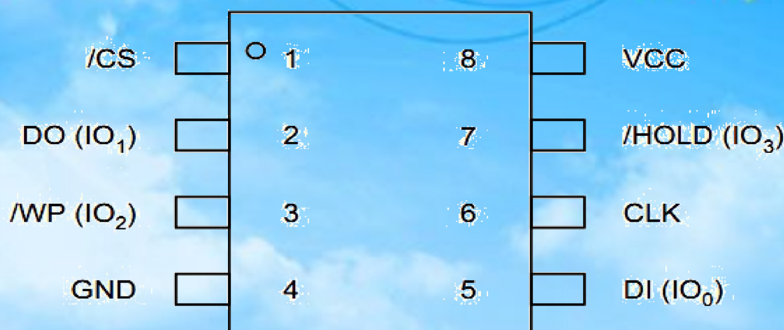


图 15.2- 2Flash 存储芯片封装

表 15.2- 2Flash 存储芯片引脚介绍

引脚号	引脚名称	输入输出（I/O）	功能
1	/CS	I	芯片选择
2	DO	I/O	数据输出
3	/WP	I/O	写保护
4	GND		地
5	DI	I/O	数据输入
6	CLK	I	时钟
7	/HOLD	I/O	保护
8	VCC		电源

2） Flash 读数据指令

由于在 STM32+LINUX 开发板中对 Flash 存储芯片的操作是已经将图片数据烧入 Flash 存储器中，所以针对 Flash 存储器的操作主要是“读数据指令”操作。

在图 15.2- 3 中，“读数据指令”允许一个字节或一个以上字节被读出。拉低/CS 引脚，把 03h 通过 DI 引脚送到芯片，然后送入 24 位地址，以上数据在 CLK 上升沿被芯片采集。芯片接收完 24 位地址后，就会把相应地址的数据在 CLK 引脚下降沿从 DO 引脚送出，且高位在前。当读完这个地址的数据后，地址自动增加，然后通过引脚把下一个地址的数据送出，形成数据流。

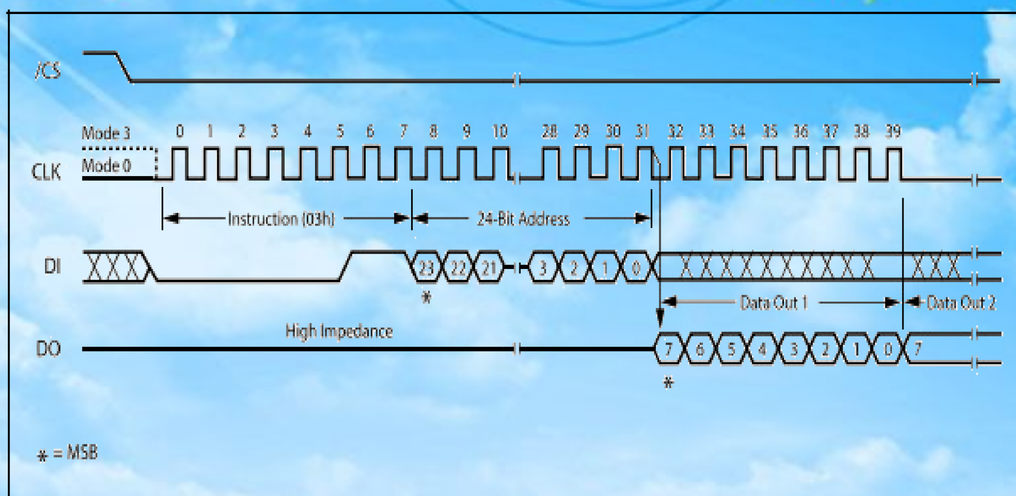


图 15.2- 3Flash 芯片读数据时序图

当数据读完后，把/CS 引脚拉高，“读数据指令”结束。当芯片在执行“页编程指令”、“擦除指令”和“读状态寄存器指令”的周期内，“读数据指令”不起作用。

26.2.4 红外编码

红外通讯技术是目前在世界范围内被广泛使用的一种无线连接技术。通过数据电脉冲和红外光脉冲之间的相互转换实现无线数据的收发。

红外通讯是以红外线作为通讯载体，在空中进行数据的传播。红外通讯由红外发射器和红外接收器组成。在红外通讯的发射端，发送的数字信号经过适当的调制编码后，再由红外发射管转变为红外光脉冲发射到空中。在红外通讯的接收端，红外接收器会对接收到的信号进行解调译码出原信号。在 STM32+LINUX 开发板中，只对红外通讯中的发射端进行操作。

红外编码首先由 9ms 的低电平和 4.5ms 的高电平构成一个引导码。紧接着的前 16 位为用户识别码，能区别不同的电器设备，防止不同机种遥控码互相干扰。芯片厂商把用户识别码固定为十六进制的一组数(两个字节)。在程序中，采用的用户识别码为 16 进制的“0x00”，用户识别反码为 16 进制“0x11”。然后就可以再通过单片机输入 16 进制操作码以及 16 进制操作反码。



图 15.2- 4 遥控信号编码波形图

如图 15.2- 5 中，以脉宽为 0.56ms、间隔为 0.56ms、周期为 1.12ms 的组合表示二进制的“0”。以脉宽为 0.56、间隔 1.68ms、周期为 2.24ms 的组合表示二进制的“1”。

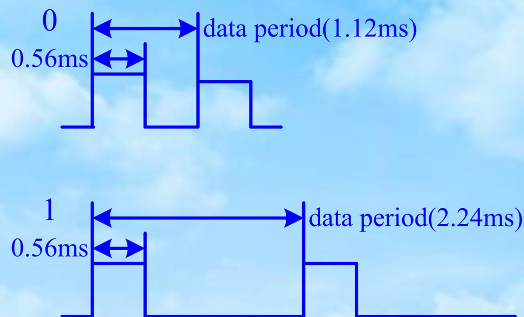


图 15.2- 5 位定义“0”和“1”

26.3 应用程序代码分析

26.3.1 网页程序

网页的代码如图 15.3- 1 所示：

```
<html><head>
<script>
function sendSer(value){
document.getElementById("ser").src="http://192.168.1.1/cgi-bin/web2ser?"+value;
}
</script>
<style type="text/css">
    body{
        margin:0 auto;
    }
</style></head><body>
<table width="1000px" align='center'>
<tr>
    <td colspan="4"><img  src='picture1.jpg' width='100%'/></td>
</tr>
<tr>
    <img id="ser">
    <td background="picture2.jpg" width="500px" height="300px">
    <input type="button" onclick="sendSer('1')" value="点亮led" /><p>
    <input type="button" onmousedown="sendSer('2')" value="关闭led" /><p>
    <input type="button" onmousedown="sendSer('3')" value="显示图片1" /><p>
    <input type="button" onmousedown="sendSer('4')" value="显示图片2" /><p>
    <input type="button" onmousedown="sendSer('5')" value="显示图片3" /><p>
    <input type="button" onmousedown="sendSer('6')" value="红外遥控" /><p>
    </td>
    <td align="right">
        <div style="margin:0 auto;">
            <iframe width="500" height="480" src="http://192.168.1.1:8080/?action=stream"/>
        </div>
    </td>
</tr></table>
</body></html>
```

图 15.3- 1 网页的代码

网页右侧显示的是摄像头拍摄画面，用 iframe 实现，源地址是"http://192.168.1.1:8080/?action=stream"这个不细说，请看“抓取摄像头图片”一

章的内容。document.getElementById 表示通过元素的 ID 特性来获取元素，例如有入下元素：<input type="text" id="button1" value="Click Me"/>那么当调用 document.getElementById("button1").Value 的时候，返回的就是"Click Me"了。看上图的中间部分的的代码，定义了六个按钮，每当有鼠标动作时，会调用 sendSer 函数，相当于根据 ID="ser" 调用 document.getElementById 函数，然后触发"http://192.168.1.1/cgi-bin/web2ser?"+value 的命令。为什么要触发这个网页命令呢，我们看下一部分内容：

26.3.2 Lua 脚本

web2ser 是一个 lua 脚本，内容很简单，如图 15.3- 2 所示：

```
#!/usr/bin/lua
io.output("/dev/ttyS0")
io.write(os.getenv("QUERY_STRING"))
```

图 15.3- 2lua 脚本

第一句作用是打开串口，os.getenv("value")：返回 value 这个进程环境变量的值，QUERY_STRING 表示截取字符串，截取调用当前文件时输入的字符串。比 io.write(os.getenv("QUERY_STRING"))则表示把获取到的字符串转化成可写入串口的值，通过串口发送出去。我们把这个脚本文件放到/www/cgi-bin/目录下，这样，当我们在浏览器地址栏中输入 192.168.1.1/cgi-bin/web2ser?abc 时，就可以在串口看到字符"abc"了。

26.3.3 单片机工程文件

单片机工程包含的文件很多，其中包括 w25q32.c(SPI_Flash)文件、R61580IC.c(LCD)文件、Irad_code.c（红外遥控）以及 led.c 文件。

w25q32.c 文件的作用是将已烧入 Flash 存储器中的图片数据通过输入相应的首地址，对图片数据进行读出操作。"读数据指令"操作在 STM32 详解篇第 8 章已经给出详细说明。

R61580IC.c 文件主要是对 LCD 液晶彩屏进行初始化配置，再将从 Flash 存储器读出的图片数据显示到 LCD 液晶彩屏上。LCD 液晶彩屏初始化配置以及图片显示请参见 STM32 详解篇第 5 章已经给出详细说明。

led.c 文件作用是点亮和关闭小灯。

着重介绍单片机工程文件的主程序、中断程序，主函数中 SystemInit 函数是对时钟的配置；SPI_FLASH_Init()函数是对 Flash 存储芯片进行引脚初始化配置；LED_Init()函数是对 led 灯配置；IR_Init()函数是对红外发射引脚的配置；R61580_init 函数是对 Lcd 液晶彩屏初始化；USART_Configuration()函数是将串口配置为 57600 波特率。



```
int main(void)
{
  uint32_t i,j;
  Delay(5000);
  Delay(5000);
  Delay(5000);
  SystemInit();
  USART_Configuration();
  SPI_FLASH_Init();
  LED_Init();
  R61580_init();
  IR_Init();
  SPI_FLASH_BufferRead_init(0);
  set_window(0,0,239,319);
  for(i=0;i<76800;i++)
  {
     Rx_Buffer[0]=SPI_FLASH_SendByte(Dummy_Byte); //240x320x2=153600,153600x8=1228800
     Rx_Buffer[1]=SPI_FLASH_SendByte(Dummy_Byte);
     WriteData((Rx_Buffer[1]<<8)|(Rx_Buffer[0])); //低字节在?
  }
  while(1)
  {
    ;
  }
}
```

图 15.3- 3 单片机工程文件主函数

上图一系列的初始化之后，将读取 flash 的指针指向第一个字节，设置好 LCD 的窗口大小，然后读取 flash 里的图片写入 LCD。

主程序中当串口发数时，产生串口接收中断，进入函数 USART2_IRQHandler()中断里（此函数在中断文件 stm32f0xx_it.c 里）。当串口发送字符‘1’时，点亮 led 灯，执行中的程序。

```
if(ch=='1')
{
   LED_Open();
}
```

图 15.3- 4 串口发‘1’程序

当串口发送字符‘3’时，执行图15.3- 5程序。由于 Flash 存储芯片中烧入的第一张图片数据的首地址为0，所以执行 SPI_Flash_BufferRead_init(0)语句。之后，利用 WriteData(Rx_Buffer[0],Rx_Buffer[1])函数将从 Flash 中读取的数据点依次写入液晶屏，这样，LCD 液晶彩屏就完整的显示了第一张图片。

```
else if(ch=='3')
{

 SPI_FLASH_BufferRead_init(0);
 set_window(0,0,239,319);
 for(i=0;i<76800;i++)
 {

    Rx_Buffer[0]=SPI_FLASH_SendByte(Dummy_Byte); //240x320x2=153600,153600x8=1228800
    Rx_Buffer[1]=SPI_FLASH_SendByte(Dummy_Byte);
    WriteData((Rx_Buffer[1]<<8)|(Rx_Buffer[0])); //低字节在?


 }
    Delay(600);
}
```

图 15.3- 5Flash 中数据显示在液晶屏程序

因为烧入 Flash 存储芯片中第二张图片的首地址为0X25800，当串口发送字符'4'的时候，执行 SPI_Flash_BufferRead_init(0X25800)程序，将 Flash 存储芯片中的第二张图片数据读出，进而显示在液晶屏上。同样，当串口发送字符 '5' 时，执行 SPI_Flash_BufferRead_init(0X4B000)，将首地址为0X4B000的第三张图片数据读出后显示在 LCD 上。

当串口发送字符'6'时，便发送一串红外编码，这个编码是任意取的值，只要红外插排可以自学习到就好。

26.4 应用程序执行

26.4.1 运行前相关配置

首先保证上位机和开发板的网络通信畅通，将单片机的程序通过 STLINK 下载到单片机里。打开 SecureCRT，启动开发板，用 WinSCP 将整个 web 文件夹放到/www 目录下：



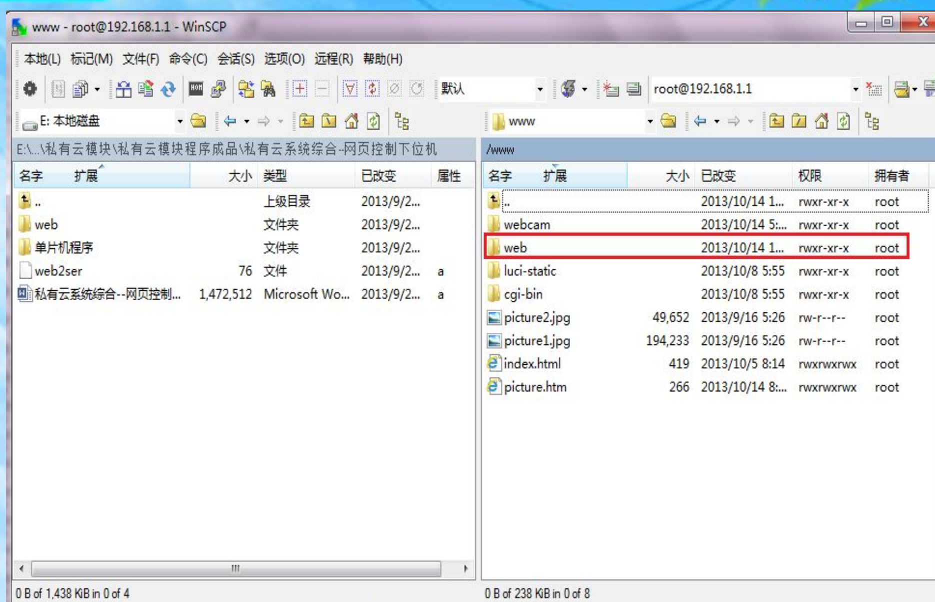


图 15.4- 1winSCP 界面

将 web2ser 文件放到/www/cgi-bin 文件夹下，插入摄像头，在终端，输入启动摄像头的命令：

mjpg_streamer -i "input_uvc.so -r 352x288 -f 15 -q 80 -y"　-o "output_http.so -p 8080 -w /www"&，&表示后台运行程序。

1）程序调试

打开火狐或者谷歌浏览器（其他浏览器不支持摄像头），在地址栏输入：http://192.168.1.1/web/car.htm，可以看到如图 15.4- 2 界面



图 15.4- 2 网页界面

图 15.4- 2 右侧显示视频，点击左侧的第 1、2 个按钮，可看见 led 灯的亮和

灭，点击接下来的三个按钮，可以显示不同的图片。点击最后一个按钮，可以进行红外遥控。比如我们的遥控接收插座，把它的接收部位对准我们的 STM32+linux 板子上的红外发射管，点击网页上的红外遥控按钮，就可以控制插座的开关啦！

26.5 项目总结

此项目比较有趣，涉及的内容也比较多，主要是利用网页编程和单片机编程来实现 Web 控制下位机操作，对于网页编程语言如 php，javascript 等，笔者还是有些余力不足呢，这里只是简单介绍一下基础，相信懂网页编程的朋友会建立出更漂亮，功能更强大的网页。

第 27 章　工业数据采集之网络虚拟串口通信

27.1 概述

这部分实现的主要功能是：用 fork 函数创建一个双进程(fork 函数在前面已经讲过了，这里不再赘述)，子进程的功能是接收上位机程序发来的 UDP 包，把 UDP 包的数据部分打印出来即输出到 SecureCRT 终端进行显示。父进程的功能是把我们输入到 buff1 里的数据(即在 SecureCRT 终端中输入的数据)打包成 UDP 包以广播包的形式发送出去。我们可以通过上位机来接收 UDP 包并将 UDP 包的数据部分转发到串口。

子进程的实现过程如图 16.1- 1：

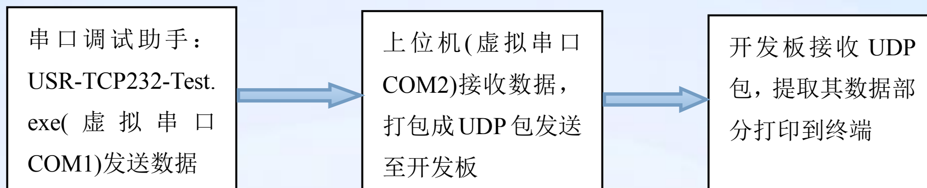


图 16.1- 1 子进程接收 UDP

父进程的实现过程如图 16.1- 2：

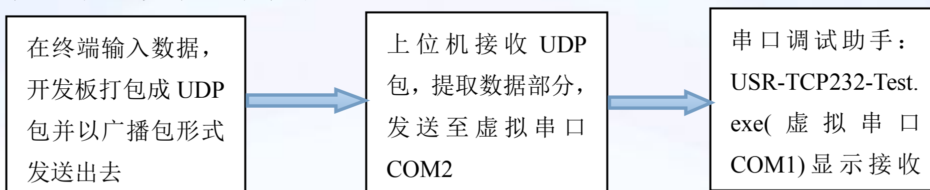


图 16.1- 2 父进程发送 UDP

这样我们就可以用串口调试助手来观察接收到的数据和发送的是否一致了。

27.2 相关知识介绍

27.2.1 虚拟串口的设置：

首先要在上位机上设置一对虚拟串口，以便在串口调试助手中观察程序运行的结果。

上位机程序中的串口和串口调试助手，我们通过虚拟串口工具将其设成一对虚拟串口，所谓的虚拟串口的含义就是并非是真实的物理串口而是我们虚拟出的一对可以相互之间进行通信的串口，通过设定虚拟串口，就可以使串口调试助手和上位机程序之间进行通信了，有关虚拟串口设定(这里以 COM1---COM2 为例进行说明)的截图如图 16.2- 1 。在图 16.2- 1 中，先选择要虚拟出的一对串口，在红框框起的 Add pair 之前进行选择，(如果要选择的串口被物理串口占用，那么可以随意选择空闲的串口进行虚拟)，选好后点击“Add pair”按钮，在左侧的“Virtual ports”(意为虚拟串口)下可以看到刚刚添加成功的一对虚拟串口 COM1 和 COM2。到此处添加虚拟串口过程完毕。

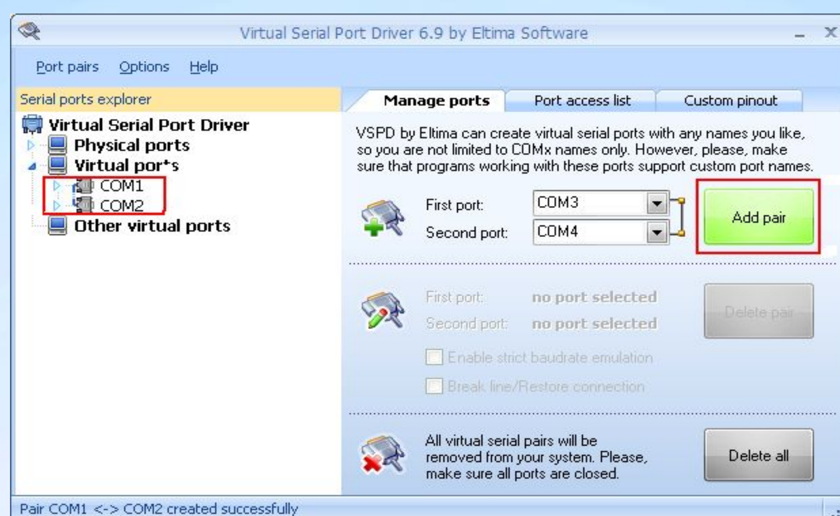


图 16.2- 1 虚拟串口界面

27.2.2 上位机软件(虚拟串口网络通信)设置：



图 16.2- 2 上位机软件

发送 UDP 数据包需要知道目的 IP 及端口号，才能把待发送的数据发送到指定计算机的指定应用程序中。如图 16.2- 2 所示：因为板子的 IP 为192.168.1.1，所以“串口发数转 UDP 包” 部分的远程 IP 就是192.168.1.1，远程端口是8003，也就是程序代码中定义的本地端口 BROADCASTPORT（此处的 BROADCASTPORT 对于上位机来说是远程端口，但对于开发板而言就是本地端口了）。知道了远程 IP 和远程端口号就可以发送 UDP 包了，本机端口可以任意设，但是不要设成电脑已经占用的端口和我们定义的电脑本地端口61557（对开发板而言是远程端口），在0~65535所有端口号中，从49152到65535，这些端口是应用程序使用的动态端口，应用程序一般不会主动使用这些端口，因此我们可以在这个区间内任意设置，这里设为60001。波特率为57600。 “收 UDP 显示至串口”部分本机端口也就是我们在程序里设置的远程端口 PORT 61557（对开发板而言是远程端口，对电脑而言就是本地端口了）。

27.3 应用程序代码分析

首先是头文件，需要包含以下头文件：

```
#include<stdlib.h>
#include<stdio.h>
#include<errno.h>
#include<string.h>
#include<netdb.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<termios.h>
#include<sys/wait.h>
#define PORT 61557                          //远程端口号
#define BROADCASTPORT 8003                  //本地端口号
#define BUF_SIZE 100                        //定义UDP数据发送长度
#define FALSE  -1
#define TRUE    0
#define DEF_IP "192.168.1.1"
```

图 16.3- 1 头文件

图 16.3- 2 是从外界接收 UDP 数据包的函数：

第一步是创建套接字，castsockfd 是套接字描述符。

第二步是配置本地信息。因为我们要接收 UDP 数据包，所以，在存放本地信息的结构体 serv_addr 中，端口号，和 IP 都要设置成本地，这里我们的端口号是 8003，是任意取值。因为 0~1024 是系统的端口号，所以取值只要大于 1024 即可。INADDR_ANY 表示使用自己的 IP 地址，memset()函数是把把 serv_addr 的所有字节设置成字符，hton()设置本地端口号 htons()表示把 16 位从主机字节序转换成网络字节序。

第三步是用 bind()函数将本地信息与套接字绑定。

第四步是接收 UDP 数据包。数据存放到 change 所指的首地址内存中。这里需要注意 revc 并不是接收数据存储区，而是接收数据的长度，这里很容易弄混淆。

最后关闭套接字，返回数据长度。

```
int udp_broadcast_service(unsigned char *change)
{
    int castsockfd, connfd;
    int optval;
    int client_len;
    int revc;
    struct sockaddr_in serv_addr, client_addr;
    unsigned short  int recv_buf[ BUF_SIZE ];
    if ( (castsockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 )  //创建socket套接字
        {
            perror("sockfd error!\n");
            exit(1);
        }
//配置本地信息，包括端口号、IP、协议族（一般为AF_INET，表示UNIX网络套接字）
//把serv_addr的所有字节设置成字符
    memset( &serv_addr, 0, sizeof(struct sockaddr_in) );
    serv_addr.sin_family = AF_INET;    //表示UNIX网络套接字
//设置本地端口号  htons()表示把16位从主机字节序转换成网络字节序
    serv_addr.sin_port = htons( BROADCASTPORT );
//INADDR_ANY表示本地ip，也就是192.168.1.1
    serv_addr.sin_addr.s_addr = htons( INADDR_ANY );
    fcntl(castsockfd,F_SETFL,O_NONBLOCK);
//调用bind函数绑定本地信息与socket相关联
    if ( bind(castsockfd, (struct sockaddr *)&serv_addr,
             sizeof(struct sockaddr)) < 0 )
        {
            perror("bind failed!\n");
            exit(1);
        }
//监听函数，监听是否有往本地端口发送数据
    client_len = sizeof( struct sockaddr_in );
//接收UDP包
    while((revc=recvfrom(castsockfd, change, BUF_SIZE, 0,
                 (struct sockaddr *)&client_addr,
                  &client_len))<=0)
          {
              ;
          }
//关闭套接字
    close( castsockfd );
//返回数据长度
    return revc;
}
```

图 16.3- 2 UDP 接收数据

图 16.3- 3 是发送 UDP 数据包的函数：

第一步是创建套接字，client_sockfd 是套接字描述符。

第二步是配置远端主机信息。因为我们要发送 UDP 数据包，所以，在存放远端主机信息的结构体 serv_addr 中，端口号，和 IP 都要设置成对方的，这里远程端口号设置成 61557。地址设置为广播地址用于发送广播包。由于在接收 UDP 数据包进程中我们已经打开了这个套接字，因此 bind()函数已经配置了，所以我们调用了 setsockopt()函数，来设定选项，关于本函数的用法请参见实战篇第一章。hton()设置对方端口号 htons()表示把 16 位从主机字节序转换成网络字节序。

第三步是将 send_buff 所指的首地址的数据，用 UDP 数据包发送出去。

最后关闭套接字，返回 0。

```c
int send_broadcast(unsigned char *send_buf)
{
    int client_sockfd;
    int optval;
    int m=0;
    struct sockaddr_in serv_addr;
    if ( (client_sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 )  //创建发送套接字
      {
         exit(1);
      }
    optval = 1;
//当一个套接字已经打开但尚未有链接时用setsocjopt()系统调用在其上设定选项
    if ( setsockopt(client_sockfd, SOL_SOCKET, SO_BROADCAST,
                (void *)&optval, sizeof(int)) < 0 )
     {
//          perror("setsockopt error!\n");
        exit(1);
     }
    memset( &serv_addr, 0, sizeof(struct sockaddr_in) );
    serv_addr.sin_family = AF_INET;                          //表示UNIX网络套接字
    serv_addr.sin_port = htons( PORT );                      //远端主机端口号
    serv_addr.sin_addr.s_addr = inet_addr("192.168.1.255"); //远程主机IP，这里设置发送广播包
    if ( sendto(client_sockfd, send_buf, strlen(send_buf), 0, //发送UDP广播包
             (struct sockaddr *)&serv_addr,
              sizeof(struct sockaddr)) < 0 )
      {
//          perror("send failed!\n");
         exit(1);
      }
    close( client_sockfd );                                  //关闭套接字
    return 0;
}
```

图 16.3- 3 UDP 发送数据

图 16.3- 4 和图 16.3- 5 是主函数：

主函数功能：用 fork 函数创建一个双进程，在两个进程中分别调用 UDP 包发送及接收子函数，实现开发板的网络通信。

```c
int main()
{
        pid_t pid,id;
        int recv_c=0;
        unsigned char buff1[100];          //发送缓冲区
        unsigned char check[100]={0};          //接收缓冲区
        pid=fork();                                   //建立新的进程。以下是两个并行的进程
        if(pid<0)
           {
               exit(1);
           }

        if(pid==0)            //子进程，接收UDP包
           {
                printf("child");
                while(1)
                {
                        recv_c=udp_broadcast_service(check);    //接收UDP包
                        printf("check=:%s\n",check);    //打印接收数据
                        usleep(10000);
                        bzero(check,100);       //清空接收缓冲区
                }
```

图 16.3- 4 主函数 1

```
			}
		else                                    //父进程，发送广播包
			{
				printf( "parent\n");
				while(1)
				 {
						id=waitpid(pid,NULL,WNOHANG);
						if(id==pid)//如果子进程死亡，父进程销毁
						  {
							printf("child exit\n");
							exit(1);
						  }

					printf("Input data");
					scanf("%s",buff1);                 //输入数据
					send_broadcast(buff1);                 //将数据以UDP包发送出去
					bzero(buff1,100);                  //清空数据缓冲区

				 }

			}
}
```

图 16.3- 5 主函数 2

27.4 应用程序执行

27.4.1 准备工作

1、首先设置一对虚拟串口 COM1-COM2(可以根据需要自行设定)，分别指定给串口调试助手和上位机程序，用于实现两者之间的串口通信。

2、要知道开发板的 IP 地址，它在上位机发送 UDP 包过程中作为远程 IP。本开发板 IP 设为 192.168.1.1。

3、将电脑与开发板用网线连接，电脑的本地 IP 需要设置成跟开发板一个网段的，　否则不能与开发板通信。本地 IP 设为 192.168.1.*，*从 2~255 之间均可。设置本地 IP 的方法详见 STM32+LINUX 开发板功能篇第三章第二节。

27.4.2 应用程序执行结果

首先，在 SecureCRT 里改变程序的执行权限，再执行程序，如

图 16.4- 1 所示。

```
root@MicroCloud:/# chmod 777 udp_sends
root@MicroCloud:/# ./udp_sends
parent
Input data
```

图 16.4- 1 执行 udp_sends

可以看到程序已经在运行了，正在等待输入数据，之后我们打开“虚拟串口



网络通信”软件并进行相应配置后点击“运行”按钮：



图 16.4- 2 虚拟串口网络通信

最后打开串口调试助手就可以进行收发数据的测试了！这里通过串口调试助手向开发板发送“123456789”，在 SecureCRT 里输入“987654321”，串口调试助手接收。串口调试助手部分截图如图 16.4- 3。

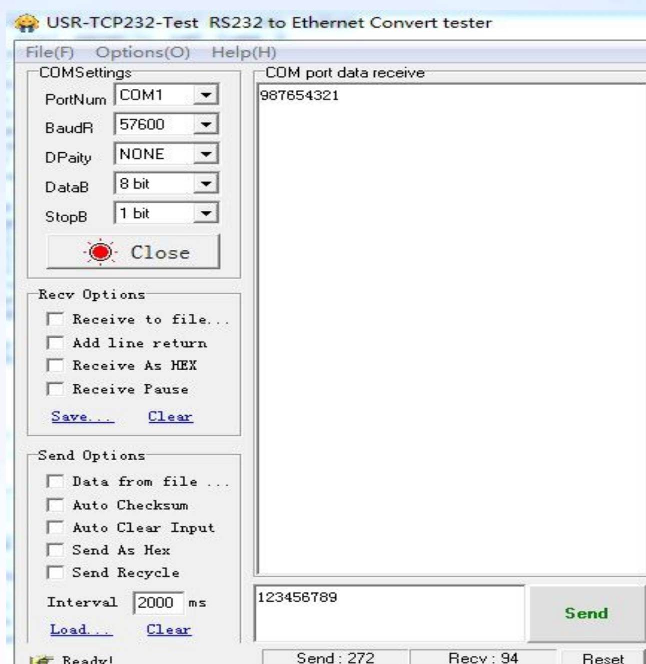


图 16.4- 3串口调试助手部分截图

SecureCRT 软件显示开发板数据接收：

```
Input datachild
check=:123456789
```

SecureCRT 软件上输入数据发送：

```
987654321
Input data
```

27.5 项目总结

本章涉及的内容主要有虚拟串口，网络通信和串口编程，这些知识在前几章都有所提及，所以本章的内容也比较容易弄懂，如果有不明白的地方可以参看第十一章的内容。

第 28 章 人机交互之远程网络显示

28.1 概述

开发板上的的彩色液晶屏可以显示任意 240*320 以内像素的图片。但单片机的内存不足以存放 240*320 个像素点的图片数据信息。像 STM32 的工程一样，将图片存放在 Flash 存储器中，然后再显示在液晶屏上当然也可以，但是每次都要烧写 Flash 难免会麻烦，那么我们怎样才能在 LCD 上显示任意的图片呢？

由前面介绍过的知识我们知道，中央模块和单片机是通过串口相连，而图片文件也可以用 WinSCP 工具轻松上传至开发板，因此，我们想办法先将图片信息转化为容易读写的二进制文件，用 WinSCP 上传至开发板，然后在利用串口将二进制文件(.bin)发送至单片机，单片机收到数据后再写入 LCD 液晶屏，从而达到显示任意图片的目的。

本章项目主要包括一个单片机工程 lcd_uart.uvproj，用于接收串口发送的数据并显示到 LCD 上；一个 C 语言应用程序 serial_send_bin.c，用于读取二进制文件（.bin）中的数据并将其发送至串口。

28.2 相关知识介绍

28.2.1 Image2TFT 取模工具的使用

Image2TFT 是一款工具软件，它可以把各种格式的图片转换成特定的数据格式以匹配单片机系统所需要的显示数据格式（比如.bin）。Image2TFT 支持的输入图像格式包括： BMP, WBMP, JPG, GIF, WMF, EMF, ICO 等。Image2TFT 的输出数据类型包括定制的二进制类型、C 语言数组类型和标准

的 BMP 格式、WBMP 格式。Image2TFT 能可视调节输入图象的数据扫描方式、灰度(颜色数)、图像数据排列方式、亮度、对比度、等等。对于包含了图像头的数据保存的图像数据文件，Image2TFT 能重新打开作为输入图像。

下面我们介绍一下如何将图片转化二进制文件：

打开 Image2TFT(在资料包中的"所用到的工具"文件夹里),如图 17.2- 1 所示：

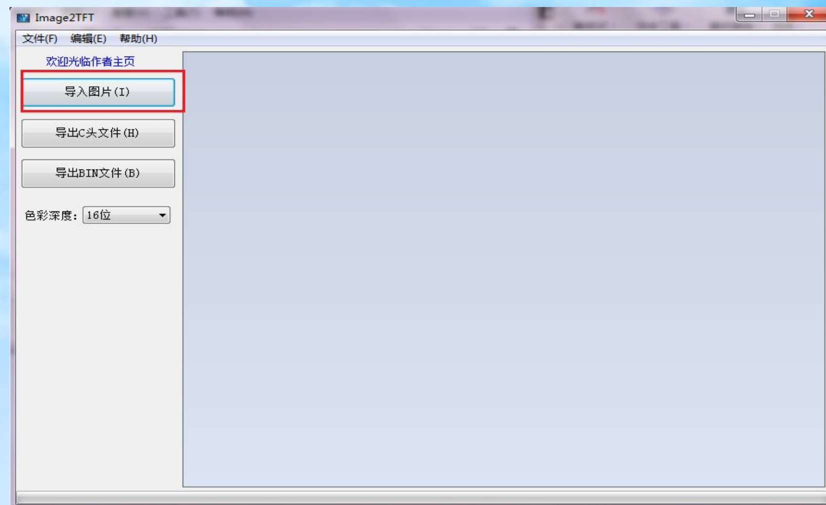


图 17.2- 1 Image2TFT 界面

点击导入图片，将所要转换的图片导入进来,导入后效果如图 17.2- 2：

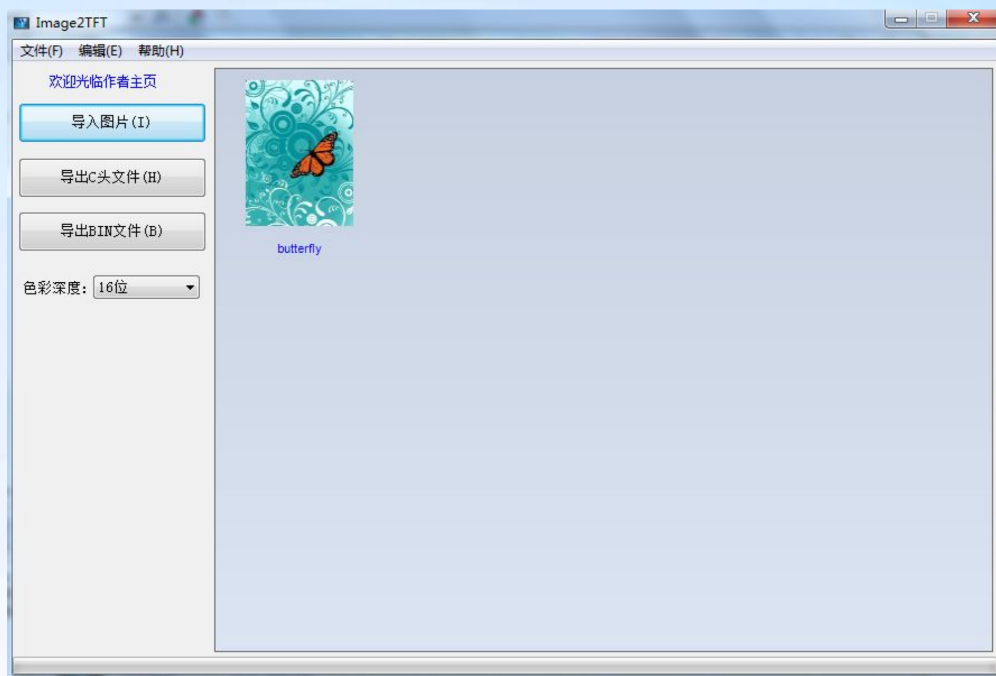


图 17.2- 2 图片导入

在菜单栏里选择编辑->调整大小，因为液晶屏显示的点数为 240*320，所以水平和垂直的点数要修改成240 和320。具体调整方式如图 17.2- 3 所示：

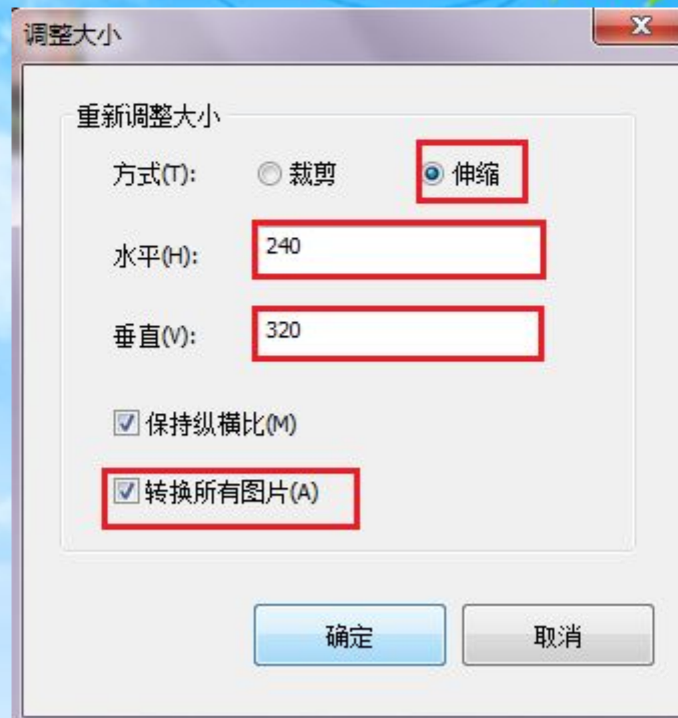


图 17.2- 3 调整大小

然后我们点击"导出 BIN 文件"，如图 17.2- 4 ：

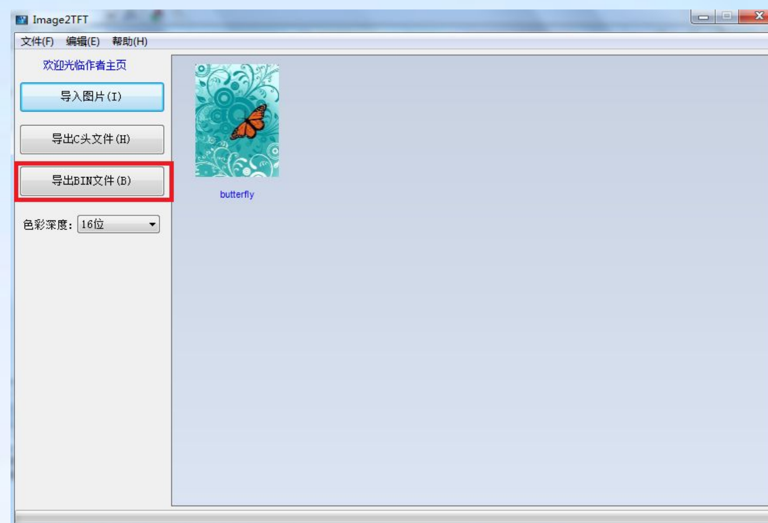


图 17.2- 4 导出图片

将二进制文件重命名(这里重命名为 butterfly.bin)，然后保存,如图 17.2- 5 所示：

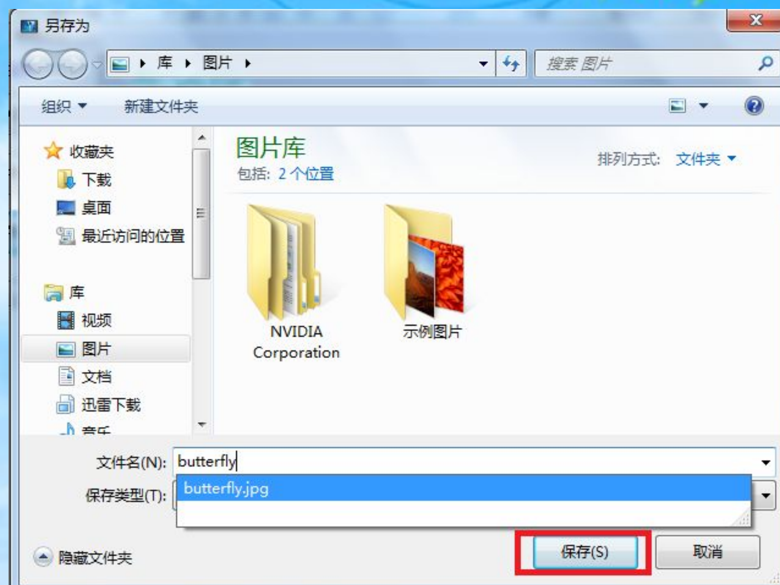


图 17.2- 5 保存图片

这样我们就在指定的文件夹里看到我们生成的，适合单片机读写的二进制文件了。文件的大小为 153.600k（图片是 240*320 个点，每一个点用两个字节表示，所以是 2*240*320=153600）。

用专门查看二进制文件的工具（010Editor V3.1.2 在资料包里可以找到）打开此二进制文件可以看到：

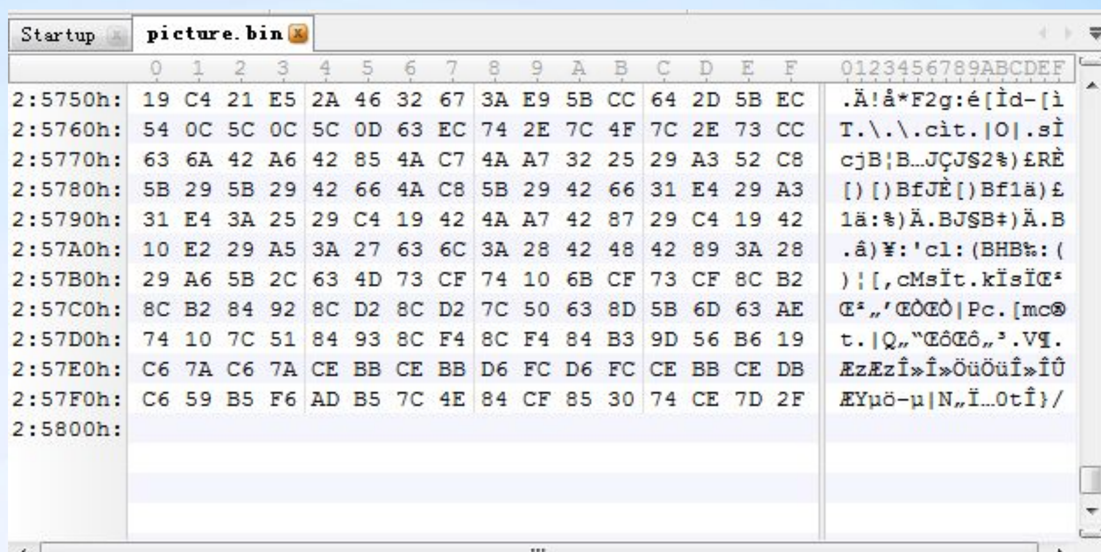


图 17.2- 6 查看二进制文件

文件的结尾为十六进制 25800h，正好是 153600 个点。

28.2.2 LCD 液晶彩屏

参看功能篇第 2 章 2.2 节

28.2.3 Linux 下文件操作编程简介

参看实战篇第 12 章第 12.2 节

28.3 应用代码分析

28.3.1 单片机工程

在 LCD 液晶彩屏上显示任意图片的单片机工程是 lcd_uart.uvproj 工程。工程的主函数如图 17.3- 1 所示，其中，函数 SystemInit()、函数 USART_Configuration()以及函数 R61580_init()都是对系统和液晶屏进行配置和初始化。对 LCD 液晶彩屏的配置及初始化在 STM32 文档里已经介绍过。工程中除了对液晶屏初始化外，主要实现串口接收图片数据显示到液晶屏。

```
int main(void)
{

  Delay(5000);
  Delay(5000);
  Delay(5000);
  SystemInit();
  USART_Configuration();
  R61580_init();                    // 液晶显示器初始化
  disp_single_color(0xf800);
  while(1)
  {
    ;
  }
}
```

图 17.3- 1 LCD 液晶彩屏任意显示图片主函数

在 lcd_uart.uvproj 单片机工程中，图 17.3- 2 中程序实现的是串口接收图片数据。当串口接收到数据时，主程序就进入串口中断，将接收到的图像数据显示到液晶屏上。在工程中，我们还加了帧头定义，用于预防干扰。由于串口传输数据有时会发生干扰现象，导致了图片数据发生错位。为了避免这种现象的发生，我们加入了帧头定义。帧头是一个数组字符串“UUU”因为在 C 语言里字符串数组的结尾默认加了一个\0，所以实际上的帧头就是“UUU\0”，转换为16进制就是0x55、0x55、0x55、0x00。



```
void USART2_IRQHandler(void)
{
    static uint8_t num=0,flag=0,buf[4]={0x00};
    static uint32_t i=0;
  if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
  {
    buf[num] = USART2->RDR;    // 从RXFIFO中读取接收到的数据
    if((buf[0]=='U')&&(flag==0))
    {
      flag=1;
      num=0;
    }
    if(flag==1)
    {
      num++;
      if(num==4)
      {
        num=0;
        if((buf[0]=='U')&&(buf[1]=='U')&&(buf[2]=='U')&&(buf[3]==0x00))
        {
          flag=2;
          set_window(0,0,239,319);                // 设置GRAM坐标
          GPIO_ResetBits(CS_PORT, CS_PIN  );       //cs=0
          GPIO_SetBits(RS_PORT, RS_PIN );         //rs=1
        }
        else
          flag=0;
      }
    }
    else if(flag==2)
    {
      num++;
      if(i<76800)
       {
         if(num==2)
            {
             WriteWord(buf[1]<<8|buf[0]);
             num=0;
             i++;
             }
       }
      else
      {
        GPIO_SetBits(CS_PORT, CS_PIN  );      //cs=1
        i=0;
        num=0;
        flag=0;
      }
    }
  }
  return;
}
```

图 17.3- 2 实现串口接收图片数据程序

28.3.2 Linux 下 C 程序

C 程序的主要功能是打开图片的二进制文件（.bin），把图片信息数据读取出来，并通过串口将数据发送出去。程序开始还是预留 10s 来切换跳帽，然后是对串口的配置，详情请看第十一章串口网络编程。这里我们主要介绍一下主函数：

```
int main()
{
    int fd;
    char *dev  = "/dev/ttyS0",file[100];
    unsigned char c[51200],buff[]={"UUU"};

    printf("please input the picture file(.bin):");
    scanf("%s",file);
    fd = OpenDev(dev);
    set_speed(fd,115200);
    if (set_Parity(fd,8,1,'N') == FALSE)  {
        printf("Set Parity Error\n");
        exit (0);
    }

    FILE *fp;
    if((fp=fopen(file,"rb+"))==NULL)
    {
        printf("can not creat file!\n");
    }
    rewind(fp);
    write(fd,buff,sizeof(buff));
    sleep(2);
    fread(c,51200,1,fp);
    while(!feof(fp))
    {
        write(fd,c,sizeof(c));
        fread(c,51200,1,fp);
    }

    close(fd);
    fclose(fp);
}
```

图 17.3- 4 主函数

首先是打开串口 ttyS0(与单片机相接的串口)，然后打印一串字符“please input the picture file(.bin):”提示要输入需要读写的二进制文件名。

scanf（“%s”，file）；程序会停在这里，等待用户输入文件名，并把文件名存入以 file 为首地址的内存里，图 17.3- 5 是配置串口速率、校验位和停止位等：

```
set_speed(fd,115200);
if (set_Parity(fd,8,1,'N') == FALSE)  {
    printf("Set Parity Error\n");
    exit (0);
}
```

图 17.3- 5 配置波特率等



fopen()是打开文件，rewind（fp）是将文件指针移动到文件的开头，fread（c，51200,1，fp）函数是从 fp 所指的文件里读 51200 个字节，读到 c 数组里，一共读一次。While 循环是判断文件是否是读到结尾了，如果没有读到结尾，就把读到的数据用串口发送出去，并继续读剩余数据；如果文件已经到结尾了，则跳出循环，最后是关闭串口和文件，程序结束。

28.4 应用程序执行

28.4.1 运行前相关配置

首先用 Image2TFT 将图片转化成二进制文件，名字可以任意取(这里以 picture.bin 为例)，然后用 WinSCP 将 picture.bin 和编译 serial_send_bin.c 形成的 serial_send_bin.o（一定要修改权限）文件上传至开发板的同一目录下，我们这里都上传至 Program 文件夹下：

```
root@MicroCloud:/Program# ls
helloworld.o          output2.bin           serial_send_bin.o
lftp.sh               picture.bin           serial_send_bin_ttyUSB0.c
output1.bin           serial_file_ftp.o     udp_serial.o
root@MicroCloud:/Program#
```

图 17.4- 1 目录下的二进制文件

打开单片机工程 lcd_uart.uvproj，将程序下载到单片机里。

28.4.2 应用程序执行结果

打开 SecureCRT 终端，进入 Program 目录下，输入执行程序命令如下：

```
root@MicroCloud:/Program# ./serial_send_bin.o
please input the picture file(.bin):
```

执行程序后会提示输入二进制文件的名字，我们输入文件名 picture.bin（注意不要输错，否则就要退出程序重新运行），

此时就可以看到 LCD 液晶屏上出现相应的图片了。

若想要更换显示的图片，则需要将图片用 Image2TFT 工具转化为二进制文件 picture.bin，上传至与 serial_send_bin.o 同一目录下，重新运行 serial_send_bin.o 就可以看到新的图片了：



图 17.4- 2 显示的图片

28.5 项目总结

总体来说，本章主要是设计一个 C 语言编程程序，和一个单片机写液晶屏的程序，内容比较少，操作也比较简单，都是前几章涉及到的知识。相信大家学到本章的时候，操作学习板已经游刃有余了。

第 29 章 家庭信息服务中心构建

29.1 概述

随着社会的不断发展，私人信息服务越来越受到大众的关注，私人信息服务包括很多种，这里我们以最常见的私人信息服务--博客作为切入点，介绍如何在在我们的 STM32+LINUX 开发板上挂载博客。

29.2 挂载博客过程

挂载博客需要安装 php,mysql,Lighttpd 及其组件，这些在 STM32+LINUX 开发板上已经安装好了，并且 php 及 Lighttpd 也已经配置完毕，接下来介绍一下数据库的配置！

29.2.1 配置 mysql

1. 在根目录下输入如下命令创建数据库文件夹：
   mkdir /mnt/data /mnt/data/mysql /mnt/data/tmp

2. 输入如下命令创建默认的数据库：
   /usr/bin/mysql_install_db --force

3. 输入如下命令启动 mysql：
   /etc/init.d/mysqld start

4. 输入如下命令创建 mysql 的密码(这里要注意 root 用户密码默认为空)
   mysql -u root -p　　#会出现输入密码的提示，输入密码后进入 mysql>
   界面如图 18.2- 1：

```
root@OpenWrt:/# /etc/init.d/mysqld start
root@OpenWrt:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.68 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
```

图 18.2- 1 mysql>界面

mysql> create database myblog;　　#创建一个叫 myblog 的数据库

mysql> show databases;　　　　　　#显示数据库信息

执行上述命令后，出现显示数据库信息界面，如图 18.2- 2：

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| myblog             |
| mysql              |
| test               |
+--------------------+
4 rows in set (0.06 sec)
```

图 18.2- 2 显示数据库信息界面

可以看到,上面是我创建的数据库 myblog，用户名：root，密码为空(下面会用到)

“Ctrl+C” -->退出数据库 出现如图 18.2- 3 所示界面：

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| myblog             |
| mysql              |
| test               |
+--------------------+
4 rows in set (0.06 sec)

mysql> ^CCtrl-C -- exit!
Aborted
root@OpenWrt:/#
```

图 18.2- 3 退出数据库

到此处数据库的配置就全部完成了！我们在博客中的文章及微语等信息就存储在数据库的 myblog 中！

29.2.2 应用执行过程及结果

- 输入如下命令启动 mysql：/etc/init.d/mysqld start
- 输入如下命令启动 Lighttpd：/etc/init.d/lighttpd start

界面如图 18.2- 4 所示：

```
root@openwrt:/# /etc/init.d/mysqld start
root@openwrt:/# /etc/init.d/lighttpd start
root@openwrt:/#
```

图 18.2- 4 显示界面

➢ 在/www 目录下新建 123.php 文件(网页文件)，内容如下：

```
<?
 echo"hello"
?>
```

接下来在浏览器中输入网址：http://192.168.1.1:81/123.php 出现如图 18.2- 5 所示界面：



图 18.2- 5 显示界面

➢ 也可以在/mnt/www 目录下新建 info.php 文件，内容如下:

```
<?php
   phpinfo()
?>
```

或

```
<?
phpinfo()
?>
```

然后在浏览器中访问 http://192.168.1.1:81/info.php 就能看到如图 18.2- 6 所示页面：

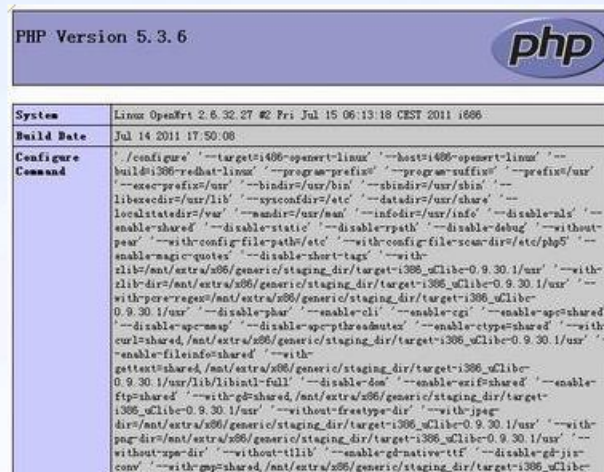


图 18.2- 6 显示界面

出现这个页面说明 php 运行环境已经搭建成功，其中显示了 php 的版本等信息！



如若没有出现预期界面，则执行/etc/init.d/lighttpd restart 命令，再重复上述过程。

接下来下载 emlog 这个博客程序：http://www.emlog.net/

下载安装包后解压，把其中的 src 文件夹放到前面所设置的 www 文件夹中，在浏览器中输入 http://192.168.1.1:81/src/

就会出现如图 18.2- 7 所示安装界面：

EMLOG
emlog 5.1.2 安装程序
MySQL数据库设置
数据库地址：127.0.0.1 (通常为 localhost，不必修改)
数据库用户名：root
数据库密码：
数据库名：myblog (程序不会自动创建数据库，请提前创建一个空数据库或使用已有数据库)
数据库前缀：emlog_ (通常默认即可，不必修改。由英文字母、数字、下划线组成，且必须以下划线结束)
管理员设置
登录名：yao
登录密码：•••••• (不小于6位)
再次输入登录密码：••••••

图 18.2- 7 安装界面

在上面输入数据库信息及个人登录信息，如上填写，之前如果没改密码，则数据库密码为空。

- 如果没有出现页面则再执行一次以下命令：

  /etc/init.d/lighttpd restart

  /etc/init.d/mysqld restart

等待约 10 秒后，安装结束，显示安装成功，如图 18.2- 8 所示：

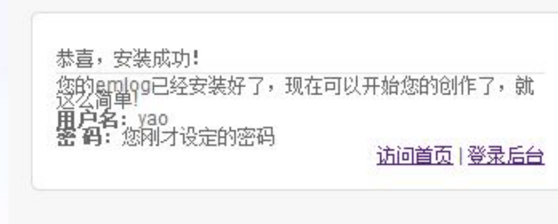


图 18.2- 8 显示安装成功

登录后的界面如图 18.2- 9 所示：

点滴记忆

使用emlog搭建的站点



首页 微语 写文章 管理站点 退出

欢迎使用emlog

作者：yao 发布于：2013-11-5 21:16 Tuesday 编辑

恭喜您成功安装了emlog，这是系统自动生成的演示文章。编辑或者删除它，然后开始您的创作吧！

评论(0) 引用(0) 浏览(0)

日历

« 2013 » « 11 »

一	二	三	四	五	六	日
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

存档

2013年11月(1)

最新评论

图 18.2- 9 登录后的界面

这就是我刚刚挂载上的博客！在这里，我们就可以写博文，发微语啦！下面就是我发的两条微语，界面如图 18.2- 10 所示：

点滴记忆

使用emlog搭建的站点



首页 微语 写文章 管理站点 退出

写微语



yao

终于完事啦！！！

2 分钟前 回复(0)



yao

今天找到工作了 很开心！

约 4 小时前 回复(0)



yao

使用微语记录您身边的新鲜事

日历

« 2013 » « 11 »

一	二	三	四	五	六	日
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

存档

2013年11月(1)

最新评论

图 18.2- 10 发的两条微语

点击图 18.2- 9 中的“管理站点”即可进入后台管理界面，进行相应的设置：(也可在浏览器中输入：http://192.168.1.1:81/src/admin/)出现如图 18.2- 11 所示界面：



图 18.2- 11 显示界面

在这里我们可以对博客进行设置，比如设置侧边栏显示信息、在首页设置导航等操作！

至此，如何在 STM32+LINUX 开发板上挂载博客的过程就全部介绍完毕！

29.3 项目总结

本章内容主要是利用 STM32+LINUX 开发板挂载私人信息服务，这里以博客为例进行了介绍，利用 SecureCRT 对开发板进行配置后，其余的基本都是上位机操作，操作简单，有兴趣的同学还可以做出拥有更棒功能、更美观界面的博客！